Gitta Puspitasari, 2012. **Algoritma *Particle Swarm Optimization* dengan *Local Search* untuk Permasalahan Penjadwalan *Permutation Flowshop***. Skripsi ini di bawah bimbingan Dr. Herry Suprajitno, M. Si, dan Drs. Eto Wuryanto, DEA. Departemen Matematika. Fakultas Sains dan Teknologi. Universitas Airlangga.

ABSTRAK

Penjadwalan merupakan aspek yang sangat penting dalam proses produksi. Hal ini dapat dilihat dari perannya dalam menentukan produk mana yang akan dikerjakan terlebih dahulu. Permasalahan penjadwalan *permutation flowshop* merupakan permasalahan *combinatorial optimization* yang dapat diselesaikan dengan menggunakan algoritma metaheuristik. Secara umum tujuan dari penjadwalan *permutation flowshop* adalah menghasilkan sebuah jadwal yang memiliki waktu penyelesaian seluruh pekerjaan atau *makespan* yang paling minimum. Skripsi ini bertujuan untuk menyelesaikan permasalahan penjadwalan *permutation flowshop* menggunakan algoritma *Particle Swarm Optimization* dengan *local search*. Proses algoritma *Particle Swarm Optimization* antara lain, membangkitkan populasi awal, membangkitkan *velocity* awal, evaluasi, menentukan *personal best* awal, menentukan *global best* awal, *update velocity*, *update* populasi, evaluasi, *update personal best*, *update global best* serta menerapkan *local search* untuk mencari solusi di daerah sekitar *global best*. Algoritma PSO kemudian diimplementasikan untuk beberapa kasus dengan menggunakan program C++ Builder, yang pertama untuk permasalahan 4-*job* 3-mesin diperoleh jadwal yaitu 2-3-1-4 dengan *makespan* sebesar 62. Selanjutnya untuk permasalahan 20-*job* 5-mesin diperoleh *makespan* sebesar 1278 dengan jadwal 3-17-15-6-9-14-7-11-19-13-1-8-5-2-4-18-16-10-20-12. *Makespan* yang didapatkan untuk permasalahan 20-*job* 10-mesin adalah 1586 dengan jadwal 5-9-12-17-15-3-18-4-2-8-19-10-6-14-20-11-13-7-1-16. Sedangkan dari hasil penyelesaian yang didapatkan untuk permasalahan 20-*job* 5-mesin dan 20-*job* 10-mesin dengan menggunakan nilai parameter *inertia weight* (w) dan *decrement factor* (α) yang berbeda-beda, menunjukkan tidak adanya hubungan antara nilai *inertia weight* yang membesar dan α mengecil dengan nilai *makespan*, atau sebaliknya.

Kata kunci : penjadwalan, penjadwalan *permutation flowshop*, *Particle Swarm Optimization*, *local search*.

Gitta Puspitasari, 2012. ***Particle Swarm Optimization* Algorithm with *Local Search* for *Permutation Flowshop Sequencing Problem***. This skripsi is supervised by Dr. Herry Suprajitno, M. Si, and Drs. Eto Wuryanto, DEA. Departement of Mathematics. Faculty of Science and Technology. Airlangga University.

ABSTRACT

Scheduling is an important aspect in production process. It can be seen from the role which products will be done first. *Permutation flowshop* scheduling problems are *combinatorial optimization* problems that can be solved by using algorithms metaheuristic. Generally, the objective of *permutation flowshop* scheduling is to produce a schedule that has a completion time of all *jobs* or the most minimum *makespan*. This skripsi aims to solve the *permutation flowshop* scheduling problems by using *Particle Swarm Optimization* algorithm with *local search*. *Particle Swarm Optimization* algorithm processing, among others, generate the initial population, generate the initial *velocity*, evaluation, determine the initial *personal best*, determine the initial *velocity*, *update* of the *global best*, *update* of the population, evaluation, *update* of the *personal best*, *update* of the *global best* and apply *local search* to find a solution in the area that around the *global best*. Then, the algorithm is implemented for the some data by using C++ Builder program. First data is problem of 4-*jobs* 3-machines, the results of completed first data by using a program, obtained schedule is 2-3-1-4 with *makespan* of 62. Futhermore, to issue 20-*jobs* 5-machines, *makespan* obtained for 1278 with schedule 3-17-15-6-9-14-7-11-19-13-1-8-5-2-4-18-16-10-20-12. *Makespan* obtained for problems of 20-*jobs* 10-machines is 1586 with schedule 5-9-12-17-15-3-18-4-2-8-19-10-6-14-20-11-13-7-1-16. While the results is obtained for problem solving 20-*jobs* 5-machines by using different parameter value of *inertia weight* (w) and *decrement factor* (α), showed isn't relation between the growing of *inertia weight* and the decreasing of *decrement factor* with the *makespan* and on the other hand.

Keywords : scheduling, *permutation flowshop* scheduling, *Particle Swarm Optimization*, *local search*.

DAFTAR ISI

Halaman

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I

PENDAHULUAN

1.1 Latar Belakang

Penjadwalan merupakan salah satu aspek yang sangat menentukan dalam proses produksi. Hal ini dapat dilihat dari perannya dalam menentukan produk mana yang akan dikerjakan terlebih dahulu. Selain itu, penjadwalan merupakan salah satu elemen perencanaan dan pengendalian produksi di bidang manufaktur dan konstruksi. Dengan adanya penjadwalan yang dilakukan secara maksimal maka biaya dan waktu untuk produksi dapat dioptimalkan. Oleh karena itu, penyusunan jadwal harus dilakukan dengan baik, dengan memperhatikan aspek-aspek yang terkait.

Permasalahan penjadwalan dalam bidang industri, terutama manufaktur menjadi menarik karena melibatkan deretan mesin yang akan digunakan untuk memproses sekumpulan operasi hingga produk-produk yang diinginkan dapat dihasilkan. Banyak permasalahan mengenai penjadwalan, misalnya pada pabrik makanan, minuman, kertas, dan pembuatan botol dimana biasanya menggunakan penjadwalan *flowshop*. Penjadwalan *flowshop* adalah menjadwalkan proses produksi dari masing-masing *n job* yang mempunyai urutan proses produksi dan melalui *m* mesin yang sama (Soetanto dan Soetanto, 1999). Dengan pengertian tersebut maka sangat memungkinkan untuk sebuah *job* melewati *job* yang lain pada saat menunggu di antrian untuk mesin yang sedang

sibuk. Mesin boleh saja tidak dioperasikan menurut prinsip *First Come First Served* (FCFS). Namun menemukan jadwal dengan waktu minimal pengerjaan untuk *job* yang diperbolehkan melewati *job* yang lain atau dengan kata lain tidak mengikuti prinsip *First Come First Served* (FCFS), lebih sulit dibandingkan dengan yang tidak diperbolehkan. *Flowshop* yang tidak memperbolehkan urutan *job* diubah antar mesin disebut dengan *permutation flowshop* (Pinedo, 2002). Fungsi tujuan yang umum dibahas pada permasalahan *permutation flowshop* adalah meminimumkan *makespan*. Jadwal dengan *makespan* yang minimum dapat meminimalkan waktu produksi.

Penjadwalan *permutation flowshop* merupakan masalah *combinatorial optimization* yang dapat diselesaikan dengan menggunakan algoritma metaheuristik, antara lain *Tabu Search*, algoritma *Simulated Anneling*, algoritma Genetika, algoritma *Ant Colony*. Seiring dengan perkembangan ilmu pengetahuan, Kennedy dan Eberhart pada tahun 1995 mengembangkan sebuah algoritma untuk menyelesaikan permasalahan *combinatorial optimization* yaitu algoritma *Particle Swarm Optimization* (PSO).

PSO merupakan salah satu algoritma yang didasarkan pada perilaku sosial di alam. PSO sendiri diinspirasi oleh interaksi sosial dan komunikasi yang terjalin antara sekawanan burung dan ikan dalam mencari makan. Pada PSO, sekawanan burung dan ikan diimplementasikan sebagai sebuah populasi yang berisi kumpulan kandidat solusi yang akan terus bergerak mendekati solusi yang optimal. Algoritma ini dimulai dengan sebuah populasi yang dibentuk secara

random. Anggota dari populasi tersebut dinamakan dengan *particle*. Tiap *particle* akan digunakan sebagai prosedur pencarian dengan saling bertukar informasi antara satu dengan yang lain tentang posisi terbaik yang ada pada ruang pencarian. Tiap *particle* akan dievaluasi untuk memperbaiki posisinya pada tiap iterasi berdasarkan dari pengalaman sebelumnya maupun pengalaman dari seluruh *particle*. Selanjutnya semua *particle* akan bergerak menuju posisi terbaik yang telah didapatkan sebelumnya.

PSO untuk permasalahan penjadwalan *permutation flowshop*, populasi yang dibentuk merepresentasikan urutan *job* yang membentuk jadwal *flowshop* pada tiap *particle*. Jadwal-jadwal tersebut akan dievaluasi dengan menggunakan kriteria *makespan* untuk mendapatkan jadwal terbaik dengan waktu minimal pengerjaan.

Local search merupakan metode untuk mengidentifikasi sebuah solusi dari suatu permasalahan dengan mempertimbangkan solusi-solusi potensial yang tersedia sampai ditemukan satu solusi yang memenuhi kriteria (Howe, 1993). *Variable Neighborhood Search* (VNS) merupakan salah satu *local search* yang bekerja pada daerah di sekitar solusi global. Berdasarkan uraian di atas, penulis tertarik untuk meminimumkan *makespan* menggunakan algoritma *Particle Swarm Optimization* dengan *local search* pada permasalahan penjadwalan *permutation flowshop*.

1.2 Rumusan masalah

1. Bagaimana mengaplikasikan algoritma *Particle Swarm Optimization* dengan *local search* untuk menyelesaikan permasalahan penjadwalan *permutation flowshop*?
2. Bagaimana mengimplementasikan algoritma *Particle Swarm Optimization* dengan *local search* pada permasalahan penjadwalan *permutation flowshop* menggunakan program C++ Builder?
3. Bagaimana mengimplementasikan program yang telah dibuat pada contoh kasus penjadwalan *permutation flowshop*?

1.3 Tujuan

1. Mengaplikasikan algoritma *Particle Swarm Optimization* dengan *local search* untuk menyelesaikan permasalahan penjadwalan *permutation flowshop*.
2. Mengimplementasikan algoritma *Particle Swarm Optimization* dengan *local search* pada permasalahan penjadwalan *permutation flowshop* menggunakan program C++ Builder.
3. Mengimplementasikan program yang telah dibuat pada contoh kasus penjadwalan *permutation flowshop*.

1.4 Manfaat

Memberikan alternatif algoritma metaheuristik untuk menyelesaikan permasalahan penjadwalan *permutation flowshop* menggunakan algoritma *Particle Swarm Optimization* dengan *local search*. Dengan demikian, algoritma tersebut dapat diimplementasikan ke dalam permasalahan yang sering terjadi pada proses produksi. Selain itu, dapat digunakan sebagai bahan pustaka di lingkungan Universitas Airlangga.

1.5 Batasan Masalah

Batasan masalah dari penulisan ini yaitu algoritma *Particle Swarm Optimization* yang digunakan merupakan algoritma yang telah dipadukan dengan menggunakan *local search,* yaitu *Variable Neighborhood Search* (VNS).

BAB II

TINJAUAN PUSTAKA

2.1 Penjadwalan

Penjadwalan memiliki pengertian secara khusus sebagai durasi dari waktu kerja yang dibutuhkan untuk melakukan serangkaian aktivitas kerja yang ada dalam bidang konstruksi (Bennatan, 1995).

Penjadwalan juga merupakan proses penyusunan daftar pekerjaan yang akan dilakukan untuk mencapai atau mewujudkan suatu tujuan tertentu yang juga memuat tabel waktu pelaksanaannya (Gould, 1997).

2.1.1 Elemen – elemen Penjadwalan

Menurut Pinedo (2002), elemen – elemen penjadwalan antara lain sebagai berikut:

1. Jumlah *job* (*n*) yang akan dijadwalkan.
2. Jumlah mesin (*m*) yang akan dilalui dalam menyelesaikan proses operasi.
3. *Processing Time* (p_{ij}), yaitu waktu proses yang diperlukan untuk sebuah mesin *j* menyelesaikan operasi dari suatu *job i*.
4. *Completion Time* (C_i), yaitu waktu penyelesaian seluruh operasi untuk suatu *job i*.
5. *Makespan*, yaitu waktu yang digunakan untuk penyelesaian seluruh *job* yang akan dijadwalkan.

2.1.2 *Gantt Chart*

Gantt chart merupakan sebuah alat sederhana yang digunakan untuk merepresentasikan waktu dimana digambarkan dengan balok atau garis yang terdapat pada sebuah tabel atau *chart* (Basu, 2008). Salah satu contoh *gantt chart* untuk permasalahan penjadwalan 3 mesin, 4 *job* ditunjukkan pada gambar 2.1 seperti berikut:

Gambar 2.1 *Gantt chart* untuk penjadwalan 3-mesin 4-*job*

2.2 *Flowshop*

Dalam bidang produksi ada kalanya terjadi sebuah kondisi dimana mesin-mesin yang berbeda diatur sebagai sebuah rangkaian dimana ada sebanyak *n job* yang tersedia akan diproses pada *m mesin* dengan urutan mesin yang sama untuk tiap *job*. Kondisi tersebut dinamakan *flowshop* (Ucar dan Tasgetiren, 2009).

Permasalahan penjadwalan *flowshop* pada dasarnya adalah untuk menemukan sebuah urutan *job* pada setiap mesin yang sesuai dengan ketentuan yang ada. Menurut T. E. Morton (1993), ada beberapa pola penjadwalan *flowshop* antara lain :

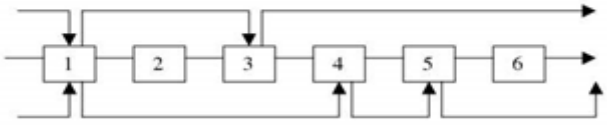
1. *Pure Flowshop*

Dalam *pure flowshop*, semua *job* memerlukan satu operasi pada tiap mesin dan dapat juga dikatakan sebagai *simple flowshop* karena tiap *job* memerlukan jumlah mesin dan urutan yang sama. Untuk memperjelas pola *pure flowshop* dapat dilihat pada gambar 2.2 seperti berikut:

Gambar 2.2 Pola *Pure Flowshop*

2. *Skip Flowshop*

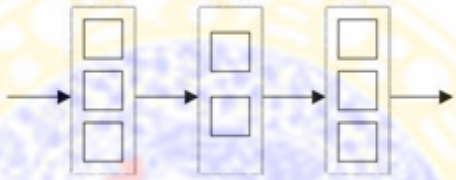
Dalam *general flowshop*, semua *job* mungkin membutuhkan kurang dari *m* operasi dan operasi-operasi tersebut mungkin tidak memerlukan pasangan mesin sesuai urutan nomornya dan mungkin tidak selalu berawal atau berakhir pada mesin yang sama. Bagi *job* yang jumlah operasinya kurang dari *m*, waktu pemrosesan untuk operasi yang bersangkutan diberi nilai nol. Pola ini disebut juga *skip flowshop* atau pola melompat, dimana mesin-mesin tertentu dapat dilompati oleh *job-job* tertentu. Untuk memperjelas pola *skip flowshop* dapat dilihat pada gambar 2.3 seperti berikut:



Gambar 2.3 Pola *Skip Flowshop*

3. *Compound Flowshop*

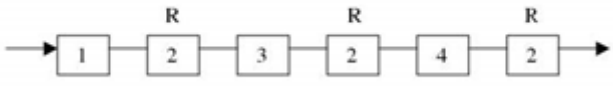
Dalam *compound flowshop*, satu mesin dalam set mesin dapat digantikan oleh sekelompok mesin. Kelompok mesin tersebut umumnya adalah mesin-mesin paralel atau jalur *batch* yang diikuti oleh mesin-mesin paralel. Contohnya, pabrik kertas, besi, pembotolan, makanan, dan lain-lain. Untuk memperjelas pola *compound flowshop* dapat dilihat pada gambar 2.4 seperti berikut:



Gambar 2.4 Pola *Compound Flowshop*

4. *Reentrant Flowshop*

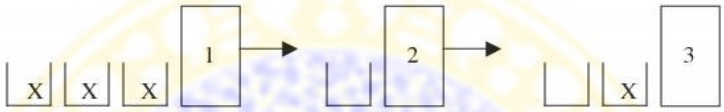
Dalam *reentrant flowshop*, beberapa mesin dapat memproses sebuah *job* lebih dari sekali. Hal ini sulit dimodelkan secara analitis, tetapi relatif mudah bila digunakan pendekatan heuristik. Contohnya, suatu pipa diekstruksi secara berulang-ulang untuk mendapatkan diameter yang lebih kecil. pada tiap ekstruksi, pipa tersebut kembali pada tungku yang sama untuk ditempa. Pola *reentrant flowshop* ditunjukkan pada gambar 2.5 seperti berikut:



Gambar 2.5 Pola *Reentrant Flowshop*

5. *Finite Queue Flowshop*

Dalam *finite queue flowshop,* ada tempat penyimpanan terbatas di depan tiap mesin. Dalam kasus khusus dapat juga terjadi bahwa tidak boleh ada antrian selain pada mesin pertama, misalnya pada industri pemrosesan logam. Untuk memperjelas pola *finite queue flowshop* dapat dilihat pada gambar 2.6 seperti berikut:



Gambar 2.6 Pola *Finite Queue Flowshop*

6. *Permutation flowshop*

Permutation flowshop termasuk dalam *pure flowshop* dimana *processing times* pada semua *job* telah diketahui sebelumnya, dengan sebuah urutan pada *n job* telah ditentukan. *Job-job* menjalankan operasi pada semua mesin yang ada tanpa mengubah urutannya.

Adapun asumsi-asumsi yang dimiliki oleh *flowshop* ini adalah :

1. Setiap *job* diproses pada semua mesin dengan urutan mesin 1,2,…,*m*.
2. Setiap mesin hanya memproses sebuah *job* pada saat yang sama.
3. Setiap *job* hanya diproses di satu mesin pada saat yang sama.
4. Operasi tidak bersifat *pre-emptable* atau sebuah *job* harus diselesaikan dulu prosesnya secara keseluruhan di sebuah mesin sebelum diproses di mesin selanjutnya.

5. *Set-up time* dari operasi termasuk pada *processing time* dan tidak bergantung pada urutan.

2.3 *Particle Swarm Optimization* (PSO)

Particle Swarm Optimization (PSO) merupakan salah satu teknik yang diinspirasi oleh perilaku sosial dan komunikasi dari gerakan kawanan hewan, seperti kawanan ikan (*school of fish*), kawanan hewan herbivor (*herd*), dan kawanan burung (*flock*) yang kemudian tiap objek hewan disederhanakan menjadi sebuah *particle*. PSO pertama kali diperkenalkan oleh Kennedy dan Eberhart pada tahun 1995. Ide awal dari *particle swarm* yang dikemukakan oleh Kennedy (seorang *social psychologist*) dan Eberhart (seorang *electrical engineer*) adalah mengarah pada pembuatan kecerdasan komputasi dengan memanfaatkan analog sederhana dari interaksi sosial, dibandingkan dengan hanya kemampuan kognitif dari individu. Kennedy dan Eberhart melakukan simulasi awal dengan meniru dari apa yang dilakukan Heppner dan Grenander dalam menganalogkan sekawanan burung pada saat mereka mencari jagung. Selanjutnya Kennedy dan Eberhart mengembangkannya ke dalam metode optimisasi yang dinamakan dengan *Particle Swarm Optimization* (PSO) (Poli dkk., 2007).

Sejumlah ilmuwan telah membuat simulasi komputer untuk interpretasi yang berbeda pada pergerakan sekawanan burung dan ikan. Reynolds, Heppner, dan Grenander menunjukkan simulasi dari kawanan burung. Reynolds tertarik pada keindahan koreografi yang dibuat oleh kawanan burung dan Heppner, seorang *zoologist*, tertarik dalam menemukan aturan mendasar yang memungkinkan sejumlah kawanan burung bergerak serentak, seringkali berubah

arah secara tiba-tiba, berhamburan, berkelompok kembali, dan lain-lain (Kennedy dan Eberhart, 1995).

Berdasarkan simulasi sederhana yang dilakukan oleh Reynolds sebelumnya, Kennedy dan Eberhart dapat menyederhanakan perilaku tiap agen dengan menyertakan "sarang" sebagai berikut:

1. Tiap agen tertarik pada lokasi sarang.
2. Tiap agen mengingat dimana posisi terdekatnya dengan sarang.
3. Tiap agen berbagi informasi kepada tetangganya (agen yang lain) tentang lokasi terdekat dengan sarang (Kennedy dan Eberhart, 1995).

2.3.1 Algoritma *Particle Swarm Optimization* (PSO)

Prosedur standar untuk menerapkan algoritma PSO adalah sebagai berikut:

1. Inisialisasi populasi dari *particle-particle* dengan *position value* dan *velocity* secara random dalam suatu ruang dimensi penelusuran.
2. Evaluasi fungsi objektif optimisasi yang diinginkan pada setiap *particle*.
3. Membandingkan nilai objektif pada tiap *particle* dengan *personal best* yang ada. Jika nilai yang ada lebih baik dibandingkan dengan nilai *personal best*, maka nilai tersebut dipakai sebagai *personal best* yang baru dan P_i sama dengan lokasi *particle* yang ada X_i dalam ruang dimensional *j*.
4. Identifikasi *particle* dalam lingkungan dengan hasil terbaik sejauh ini.
5. *Update velocity* dan posisi *particle*.

6. Kembali ke langkah 2 sampai kriteria terpenuhi, berhenti pada nilai objektif yang cukup baik atau sampai pada jumlah maksimum iterasi.

(Teugeh dkk., 2009)

2.3.2 Inisialisasi populasi

PSO dimulai dengan membangkitkan populasi secara random. Populasi berisi kandidat-kandidat solusi yang digunakan untuk menyelesaikan sebuah permasalahan yang akan terus diperbaiki pada tiap iterasinya. Pada PSO, populasi yang diambil umumnya tidak terlalu besar, antara 20 sampai 50. Untuk permasalahan *permutation flowshop* sendiri, menurut Tasgetiren (2004) ukuran populasi yang diambil sebanyak 2 x jumlah *job* dengan batas atas untuk pengambilan populasi secara acak (x_{max}) merupakan bilangan bulat positif.

2.3.3 *Personal Best* (P_i^t)

Personal best (P_i^t) merupakan vektor yang menggambarkan posisi terbaik untuk *particle i* berdasarkan nilai *fitness* terbaik hingga iterasi *t*. Fungsi objektif yang digunakan untuk meminimalkan *makespan* adalah $f(\pi_i^t \leftarrow X_i^t)$ dimana π_i^t menunjukkan permutasi *job* pada *particle* X_i^t, sedangkan *personal best* pada *particle i* ditunjukkan seperti $f(\pi_i^t \leftarrow P_i^t) \leq f(\pi_i^{t-1} \leftarrow P_i^{t-1})$. Secara sederhana, fungsi *fitness* pada *personal best* dapat ditulis $f_i^p = f(\pi_i^t \leftarrow P_i^t)$. *Personal best* dapat ditulis $P_i^t = [p_{i1}^t, p_{i2}^t, \ldots, p_{in}^t]$ untuk tiap *particle i*, dimana posisi terbaik (p_{ij}^t) direpresentasikan sebagai *position value* untuk *personal best* ke-*i* terhadap *job j* pada iterasi *t*. Pada

setiap iterasinya, *personal best* akan diperbaiki apabila $f_i^t < f_i^p$, untuk *i* = 1, 2, ..., ρ (Tasgetiren dkk., 2004).

2.3.4 *Global Best* (G^t)

Global best (G^t) merupakan *particle* terbaik, yang berisi kumpulan posisi terbaik, dari seluruh *personal best* pada iterasi *t*. Berdasarkan pengertian tersebut, *global best* dapat diperoleh dengan $f(\pi^t \leftarrow G^t) \leq f(\pi_i^t \leftarrow P_i^t)$ untuk *i* = 1, 2, …, ρ. Secara sederhana, fungsi *fitness* pada *global best* dapat ditulis $f^g = f(\pi^t \leftarrow G^t)$. *Global best* sendiri dapat ditulis $G^t = [g_1^t, g_2^t, \dots, g_n^t]$ dimana g_j^t merupakan *position value* untuk *particle* yang terpilih sebagai *global best* terhadap *job j* pada iterasi *t*. Pada setiap iterasinya, *global best* akan diperbaiki apabila $f_l^t < f^g$, untuk *i* = 1, 2, ..., ρ (Tasgetiren dkk., 2004).

2.3.5 *Update Velocity* (v_{ij}^t)

Particle dapat bergerak menelusuri ruang solusi dengan *velocity*. Pada dasarnya *velocity* digunakan untuk menentukan arah dimana suatu *particle* diperlukan untuk berpindah dan memperbaiki posisi sebelumnya sehingga *particle* dapat menuju ke ruang solusi yang lebih baik. *Velocity* akan diperbaiki pada tiap iterasi dengan memperhatikan beberapa hal, antara lain *velocity* sebelumnya, pengaruh *personal best* dan *global best* pada iterasi sebelumnya. *Velocity* diperbaiki dengan menggunakan persamaan berikut:

$$v_{ij}^t = w^{t-1} v_{ij}^{t-1} + c_1 r_1 \left(p_{ij}^{t-1} - x_{ij}^{t-1}\right) + c_2 r_2 \left(g_j^{t-1} - x_{ij}^{t-1}\right) \qquad (2.1)$$

Selanjutnya akan dibentuk populasi baru dengan memperbaiki *position value* pada tiap *particle* dengan persamaan berikut :

$x_{ij}^{t} = x_{ij}^{t-1} + v_{ij}^{t}$, dimana *i* = 1, 2, …, ρ ; *j* = 1, 2, …, *n* (2.2)

Dengan

v_{ij}^{t} : *velocity* yang terletak pada *particle i, job j* untuk iterasi *t*-1

w^{t-1} : *inertia weight* pada iterasi *t*-1

x_{ij}^{t} : *position value* yang terletak pada *particle i, job j* untuk iterasi *t*

x_{ij}^{t-1} : *position value* yang terletak pada *particle i, job j* untuk iterasi t-1

p_{ij}^{t-1} : *personal best* yang terletak pada *particle i, job j* untuk iterasi *t*-1

g_{j}^{t-1} : *global best* pada *job j* untuk iterasi *t*-1

c_1 dan c_2 : *cognitive* dan *social parameter*

r_1 dan r_2 : random uniform [0,1]

(Tasgetiren dkk., 2004)

2.3.5.1 *Inertia Weight* (w^t)

Inertia weight merupakan salah satu parameter yang ada pada PSO yang berfungsi sebagai pengontrol pengaruh dari *velocity* sebelumnya untuk *velocity* yang sekarang. Pada dasarnya, *inertia weight* diperkenalkan untuk keseimbangan antara kemampuan penelusuran *global* dan *local*. *Inertia weight* akan diperbaiki dengan menggunakan persamaan:

$w^t = w^{t-1} \times \alpha$ (2.3)

Dengan

w^t : *inertia weight* pada iterasi *t*

w^{t-1} : *inertia weight* pada iterasi *t*-1

α : faktor pengurangan (*decrement factor*)

Nilai awal pada parameter ini biasanya berkisar antara 0.4 sampai dengan 0.9 dan nilai α yang biasa digunakan 0.975 (Uysal dan Bulkan, 2008).

2.3.5.2 *Cognitive* dan *Social Parameter* (c_1 dan c_2)

Cognitive parameter merupakan parameter yang digunakan untuk mengontrol pengaruh dari *personal best* terhadap *position value* pada iterasi sebelumnya. *Social parameter* merupakan parameter yang digunakan untuk mengontrol pengaruh dari *global best* terhadap *position value* pada iterasi sebelumnya. *Cognitve* dan *social parameter* berisi konstanta-konstanta yang umumnya bernilai 1.5 – 2.0 dan 2.0 – 2.5 untuk masing-masing parameter (Hasan, 2004).

2.4 *Smallest Position Value* (SPV) *rule*

Dalam permasalahan penjadwalan *permutation flowshop*, SPV *rule* merupakan sebuah aturan yang digunakan untuk mendapatkan permutasi *job*. SPV memiliki aturan yang sederhana yaitu dengan mengurutkan nilai yang paling kecil hingga nilai yang terbesar. Nilai yang dimaksud adalah *position value* (x_{ij}) yang berada pada tiap *particle*. Dimana setiap nilai yang diurutkan mewakili *job* yang akan dikerjakan.

Tabel 2.1 Representasi SPV *rule*

Dimension (*j*)	**1**	**2**	**3**	**4**	**5**	**6**
Position x_{ij}	1.8	-0.99	3.01	-0.72	-1.20	2.15
Job π_{ij}	5	2	4	1	6	3

Dari tabel di atas dapat dilihat bahwa *job* 5 dikerjakan terlebih dahulu karena *position value* pada *job* 5 bernilai paling kecil dibandingkan dengan yang lain. Selanjutnya yang dikerjakan adalah *job* 2 yang memiliki nilai terkecil kedua setelah *job* 5. Begitu seterusnya sampai didapatkan permutasi *job* yaitu 5-2-4-1-6-3.

2.5 *Local Search*

Metode *local search* untuk *combinatorial optimization* adalah dengan melakukan serangkaian perubahan yang terjadi di sekitar solusi awal yang akan meningkatkan nilai fungsi objektif hingga optimum lokal ditemukan (Mladenovic dan Hansen, 1997). Beberapa *local search* antara lain :

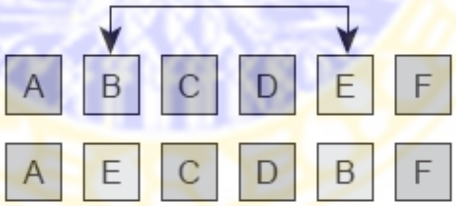
1. *Variable Neighborhood Search* (VNS)

VNS berasal dari ide yang sederhana dan efektif, yaitu dengan mengubah sistematika pada *neighborhood* ke dalam algoritma *local search*. Berbeda dengan metode *local search* yang lain seperti II dan VND, VNS akan mengeksplorasi lebih jauh *neighborhood* yang berasal dari solusi saat ini sehingga dimungkinkan untuk mendapatkan solusi yang lebih baik. Menurut Mladenovic dan Hansen (1997), VNS didasarkan pada tiga prinsip yaitu:

a. Minimum lokal antara satu *neighborhood* dengan yang lain belum tentu sama.
b. Minimum global merupakan minimum lokal untuk semua kemungkinan susunan *neighborhood*.
c. Untuk beberapa masalah, minimum lokal terhadap satu atau beberapa *neighborhood* relatif dekat satu sama lain.

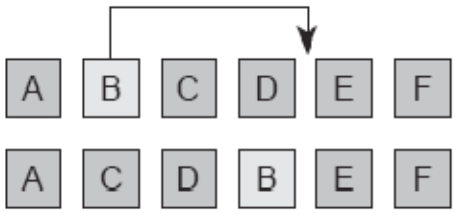
Solusi yang diperoleh sebelumnya dapat bergerak dengan operasi *neighborhood*. Pada VNS terdapat dua operasi *neighborhood* yang dapat digunakan yaitu :

1) *Interchange* adalah fungsi yang digunakan untuk bergerak dimana dua operator yang dipilih secara acak dan kemudian ditukar. Misalnya dapat dilihat pada gambar 2.7, B dan E dipilih secara acak kemudian ditukar.



Gambar 2.7 Contoh fungsi *interchange*

2) *Insert* adalah fungsi yang digunakan untuk menyisipkan sebuah operator yang dipilih secara acak di depan atau dibelakang operator lain yang dipilih secara acak pula. Misalnya dapat dilihat pada gambar 2.8, B dan E dipilih secara acak maka B disisipkan di depan E.



Gambar 2.8 Contoh fungsi *insert*

2. *Iterative Improvement* (II)

Pada *iterative improvement* diawali dengan menentukan kandidat solusi terlebih dahulu kemudian dipilih solusi di sekitar kandidat tersebut sampai diperoleh local minimal.

3. *Variable Neighborhood Descent* (VND)

Pada *variable neighborhood descent* diawali dengan menentukan kandidat solusi kemudian dicari himpunan solusi-solusi di sekitar kandidat tersebut, dicari solusi terbaiknya. Hal ini diulangi hingga iterasi yang dikehendaki.

2.6 C++ Buider

C++ Builder adalah suatu alat pengembangan aplikasi (*development tool*) berbasis Microsoft Windows yang menerapkan konsep visualisasi. Dengan adanya dukungan visualisasi ini C++ Builder menjadi mudah digunakan untuk membuat aplikasi-aplikasi secara cepat. Dalam pengeksekusian kode programnya C++ Builder menerapkan konsep *event-driven*, yaitu pengeksekusian yang didasarkan atau kejadian (*event*) tertentu. Setiap kejadian tersebut memiliki kode program tersendiri yang disimpan dalam sebuah fungsi.

Secara umum lingkungan yang terdapat dalam C++ Builder atau yang disebut IDE (*Integrated Development Environment*) dibagi menjadi lima bagian besar, antara lain:

1. *Main Window*, yang terdiri dari:
   a. *Main Menu*
   b. *Tools Panel*
   c. *Component Pallete*
2. *Object Treeview*

   Object treeview akan menampilkan daftar komponen (baik visual maupun non-visual) yang ditempatkan di dalam *form*, *data module*, maupun *frame*. Yang kemudian komponen-komponen tersebut ditampilkan dalam bentuk *tree* yang menunjukkan hubungan logik antara komponen *parent* dan komponen-komponen yang terdapat di dalamnya.
3. *Form Designer*

   Form designer adalah bagian yang digunakan untuk membuat *form* yang kemudian akan ditampilkan dalam aplikasi.
4. *Object Inspector*

   Object inspector digunakan untuk mengatur dan melakukan pengesetan terhadap properti dan *event* suatu objek dari C++ Builder.
5. *Code Editor*

   Code editor berfungsi untuk menyunting atau menuliskan kode-kode program yang akan digunakan sebagai pengontrol aplikasi yang dibuat dalam C++ Builder.

Komponen-komponen dasar yang sering digunakan antara lain:

- *#include*<namafile>

  #include menginstruksikan kepada kompiler untuk menyisipkan file lain saat program dikompilasi.

- Pernyataan

  Pernyataan berupa instruksi untuk memerintahkan komputer melakukan sesuatu.

- *TLabel*

  Komponen ini berfungsi untuk mencetak satu baris teks di dalam sebuah *form.*

- *TButton*

  Komponen ini berfungsi untuk menambahkan tombol di dalam sebuah *form.*

- *TEdit*

  Komponen ini berfungsi sebagai penerima input yang dilakukan.

- Pernyataan *if*

  Pernyataan *if* digunakan untuk mengambil keputusan berdasarkan suatu kondisi. Kadangkala pernyataan *if* disertai dengan pernyataan *else* untuk suatu keputusan pada kondisi yang berlawanan.

- Pernyataan *for*

  Pernyataan *for* digunakan untuk mengulang pengeksekusian terhadap satu atau sejumlah pernyataan.

(Heryanto dan Budi, 2006)

BAB III

METODE PENELITIAN

Langkah – langkah penyelesaian permasalahan penjadwalan *permutation flowshop* menggunakan algoritma *Particle Swarm Optimization* (PSO) dengan *local search* adalah sebagai berikut :

1. Melakukan tinjauan pustaka tentang permasalahan penjadwalan *permutation flowshop,* algoritma *Particle Swarm Optimization* (PSO) serta *local search.*
2. Prosedur algoritma *Particle Swarm Optimization* (PSO) sebagai berikut :
   a. Inputkan banyak *job* (*n*), banyak mesin (*m*), *processing time* (p_{ij}), max iterasi, x_{max}, v_{max}, dan banyaknya anggota populasi (ρ), dimana $\rho = 2 \times n$.
   b. Inisialisasi parameter – parameter yang terkait, yaitu c_1, c_2, w^0, $\propto$.
   c. Mulai iterasi *t* = 0, lakukan langkah – langkah sebagai berikut :
      i. Inisialisasi populasi dengan membangkitkan *particle* X_i^0 secara random [0, x_{max}) dimana $X_i^0 = [x_{i1}^0, x_{i2}^0, .., x_{in}^0]$ untuk *i* = 1, 2, …, ρ.
      ii. Inisialisasi *velocity* dengan membangkitkan V_i^0 secara random $[-v_{max}, v_{max})$ dimana $V_i^0 = [v_{i1}^0, v_{i2}^0, .., v_{in}^0]$ untuk *i* = 1, 2, …, ρ.
      iii. Permutasi *job* pada tiap *particle i,* $\pi_i^0 = [\pi_{i1}^0, \pi_{i2}^0, \dots, \pi_{in}^0]$, didapatkan dengan menggunakan SPV *rule* seperti berikut :
         - Asumsi awal *position value* (x_{ij}^t) pada tiap *particle i*, mewakili *job j* untuk *i* = 1, 2, …ρ dan *j* = 1, 2, …, n. Misal x_{i1}^t mewakili *job* 1.

- Mengurutkan *position value* dari yang terkecil hingga nilai terbesar dengan menggunakan asumsi di atas.
- Urutan *position value* dari yang terkecil menggambarkan *job j* akan dikerjakan lebih dulu.
- Dapatkan *permutasi job* untuk tiap *particle i*.

iv. Mengevaluasi tiap *particle* dengan fungsi objektif $f(\pi_i^t \leftarrow X_i^t) = C_{max}(\pi_n, m)$ sehingga diperoleh nilai objektif f_i^0, untuk *i* = 1, 2, …, ρ. Dapatkan nilai *fitness* terbaik dari tiap *particle i* untuk memperoleh *personal best*, dimana diasumsikan $f_i^p = f_i^0$ sehingga $P_i^0 = X_i^0$. Kemudian temukan *global best* $G^0 = X_l^0$ yang didapat dari *position value* dengan nilai *fitness* terbaik dari semua *particle i*, $f_l^0 = min\{f_i^0\}$ sehingga $f^g = f_l^0$.

d. Untuk *t* = *t*+1, lakukan langkah – langkah sebagai berikut :

i. Membangkitkan r_1, r_2 secara random [0, 1], kemudian *update inertia weight* dan *velocity* dengan menggunakan persamaan (2.3) dan (2.1).

Lakukan langkah berikut jika $v_{ij}^t > v_{max}$ atau $v_{ij}^t < v_{min}$

- Jika $v_{ij}^t > v_{max}$ maka $v_{ij}^t = v_{max}$.
- Jika $v_{ij}^t < v_{min}$ maka $v_{ij}^t = v_{min}$.

ii. *Particle* yang baru dibentuk dengan meng-*update position value* menggunakan persamaan (2.2).

iii. Permutasi *job* pada tiap *particle i*, $\pi_i^t = [\pi_{i1}^t, \pi_{i2}^t, \dots, \pi_{in}^t]$, didapatkan dengan menggunakan SPV *rule* seperti langkah iii pada *t* = 0.

iv. Tiap *particle* i dievaluasi dengan menggunakan fungsi objektif yang telah ada. Kemudian untuk setiap *particle i*, *personal best* diperbaiki dengan membandingkan nilai objektif pada iterasi *t* dengan nilai *fitness* untuk *personal best* pada iterasi sebelumnya dengan ketentuan sebagai berikut:

- Jika $f_i^t < {}_i^p$, i = 1, 2, …, ρ , maka *personal best* diperbaiki menjadi $P_i^t = X_i^t$ dengan $f_i^p = f_i^t$, untuk i = 1, 2, …, ρ.
- Jika $f_i^t \geq f_i^p$, i = 1, 2, …, ρ , maka *personal best* $P_i^t = X_i^{t-1}$ dengan $f_i^p = f_i^{t-1}$, untuk i = 1, 2, …, ρ.

v. Menemukan nilai minimum pada *personal best* dimana $f_l^t = min\{f_i^p\}$ untuk i = 1, 2, …, ρ dan $l \in i$, dengan ketentuan sebagai berikut:

- Jika $f_l^t < f^g$ maka *global best* diperbaiki menjadi $G^t = X_l^t$ dengan $f^g = f_l^t$.
- Jika $f_l^t \geq f^g$ maka $G^t = X_l^{t-1}$ dengan $f^g = f_l^{t-1}$.

vi. Menggunakan VNS sebagai *local search* untuk menemukan solusi yang lebih optimal dengan ketentuan sebagai berikut :

1) Mengubah *global best* yang telah didapatkan ke dalam permutasi *job* dengan menggunakan SPV *rule*.
2) Asumsikan s_0 adalah permutasi *job* (π^t) yang berada pada *global best*.
3) η, κ dipilih secara random untuk *integer* [1, n] dengan $\eta \neq \kappa$.

4) Mendapatkan permutasi *job* s yang merupakan permutasi *job* baru melalui operasi *insert* untuk permutasi *job* s_0 dengan ketentuan sebagai berikut:
   a) Jika $\eta < \kappa$ maka pindahkan *job* yang berada pada dimensi η kemudian sisipkan pada *job* yang berada di depan *job* yang berada pada dimensi κ.
   b) Jika $\eta > \kappa$ maka pindahkan *job* yang berada pada dimensi η kemudian sisipkan pada *job* yang berada di belakang *job* yang berada pada dimensi κ.
5) Atur loop = 0, kcount = 0, maxmethod = 2
6) η, κ dipilih secara random untuk *integer* [1, n] dengan $\eta \neq \kappa$. Melakukan operasi *neighborhood* dengan ketentuan sebagai berikut:
   a) Jika kcount = 0 maka dapatkan permutasi *job* s_1 dengan menggunakan operasi *insert* pada permutasi *job* s dengan ketentuan sebagai berikut:
      i) Jika $\eta < \kappa$ maka pindahkan *job* yang berada pada dimensi η kemudian sisipkan pada *job* yang berada di depan *job* yang berada pada dimensi κ.
      ii) Jika $\eta > \kappa$ maka pindahkan *job* yang berada pada dimensi η kemudian sisipkan pada *job* yang berada di belakang *job* yang berada pada dimensi κ.
   b) Jika kcount = 1 maka dapatkan permutasi *job* s_1 dengan menggunakan operasi *interchange* pada permutasi *job* s. Yaitu

dengan menukar *job* yang berada pada dimensi η dengan *job* yang berada pada dimensi κ.

7) Mengevaluasi permutasi *job* s_1, kemudian $f(s_1)$ dibandingkan dengan $f(s)$dengan ketentuan:
   a) Jika $f(s_1) < f(s)$ maka kcount = 0 dan $s = s_1$.
   b) Jika $f(s_1) > f(s)$ maka kcount = kcount+1.
8) Ulangi langkah 6) selama kcount < maxmethod, atur loop = loop+1 jika kcount $\geq$ maxmethod dan ulangi langkah 5) hingga loop < n*(n-1).
9) Membandingkan $f(s)$ dengan $f(\pi^t)$ dengan ketentuan sebagai berikut:
   a) Jika $f(s) \leq f(\pi^t)$ maka perbaiki *global best* G^t dengan permutasi *job* $\pi^t = s$.
   b) Jika $f(s) > f(\pi^t)$ maka permutasi *job* $\pi^t = s_0$.

vii. Mendapatkan penjadwalan *flowshop*.

e. Kembali ke prosedur d. jika $t <$ max iterasi. Jika $t \geq$ max iterasi maka berhenti.

3. Mengimplementasikan algoritma yang telah dibuat ke komputer dengan menggunakan bahasa pemrograman C++ Builder.
4. Menguji coba program pada contoh kasus yang telah diselesaikan secara manual (kasus kecil) dan kasus dengan ukuran agak besar.

BAB IV
PEMBAHASAN

Pada bab ini akan dijelaskan mengenai penerapan algoritma *Particle Swarm Optimization* (PSO) dengan *local search* untuk permasalahan penjadwalan *permutation flowshop*.

4.1 Prosedur Algoritma *Particle Swarm Optimization* (PSO)

Menurut Eberhart (1996), prosedur umum algoritma PSO ditunjukkan pada Gambar 4.1.

```
Prosedur umum algoritma PSO
{
  Set t←0 ;
  Initialize S and Set P ≡ S ;
  Evaluate S and P, and define index g of the best position;
  While (termination criterion not met)
  {
    Update S using equations (2.1) and (2.2);
    Evaluate S;
    Update P and redefine index g;
    Set t←t+1;
  }
  End While
  Print best position found;
}
```

Gambar 4.1 Prosedur Umum Algoritma PSO

Prosedur *initialize S* seperti pada ilustrasi Gambar 4.1 merupakan prosedur inisialisasi populasi (*swarm*) awal. Pada implementasinya, inisialisasi dapat dibagi menjadi 2, yaitu inisialisasi populasi dan inisialisasi parameter. Inisialisasi parameter digunakan untuk menginputkan nilai parameter yang diperlukan dalam algoritma PSO. Untuk penerapan PSO dalam permasalahan penjadwalan *permutation flowshop*, setelah menginisialisasi populasi dilakukan prosedur pencarian *job sequence* pada tiap *particle*. Hal ini dilakukan untuk mendapatkan permutasi *job* yang kemudian akan dievaluasi sehingga diperoleh nilai *makespan*

dari tiap *particle* tersebut. Untuk iterasi awal, *personal best* (pada Gambar 4.1 disimbolkan dengan *P*) berisi populasi awal *S*. *Define index g* merupakan prosedur untuk mendapatkan *global best*, ditunjukkan dengan *particle* yang memiliki nilai *makespan* terkecil pada populasi.

Sedangkan untuk prosedur *update S*, persamaan (2.1) dan (2.2) digunakan untuk memperbaiki *velocity* dan populasi. Proses ini akan berlangsung terus menerus sebanyak iterasi yang diinginkan. Penerapan *local search* dilakukan setelah *global best* diperbaiki, sebelum iterasi berikutnya dilakukan. *Local search* atau mutasi bersifat *optional*, yaitu apabila telah dilakukan *local search* tidak pelu dilakukan mutasi. Prosedur PSO dengan *local search* untuk permasalahan penjadwalan *permutation flowshop* dapat dilihat pada Gambar 4.2.

```
PSO dengan local search
{
        Initialize parameters;
        Initialize population;
        Find permutation;
        Evaluate;
        Do
        {
           Find the personal best;
           Find the global best;
           Update velocity;
           Update position;
           Find permutation;
           Evaluate;
           Apply local search or mutation (optional);
        }
        While (Termination)
}
```

Gambar 4.2 PSO dengan *Local Search*

Setelah didapatkan prosedur PSO dengan *local search* selanjutnya algoritma PSO dengan *local search* untuk *permutation flowshop* akan dibahas pada subbab 4.2, sedangkan program, data, dan contoh kasus permasalahan penjadwalan *permutation flowshop* yang diselesaikan secara manual masing-masing akan dibahas pada subbab 4.3, 4.4, 4.5.

4.2 Algoritma PSO dengan *Local Search* untuk *Permutation Flowshop*

Sesuai dengan ilustrasi yang disajikan pada Gambar 4.2, maka penjelasan lebih lanjut mengenai prosedur algoritma PSO dengan *local search* untuk permasalahan penjadwalan *permutation flowshop* adalah sebagai berikut :

4.2.1 Prosedur Inisialisasi Parameter

Pada algoritma PSO, parameter-parameter yang dibutuhkan tidak terlalu banyak yaitu, c_1, c_2, w^0, α. Parameter-parameter tersebut dapat dilihat pada Gambar 4.3, yang digunakan dalam proses pergerakan seluruh *particle* pada tiap iterasinya.

```
Inisialisasi Parameter
{
  c1      ;      // cognitive parameter
  c2      ;      // social parameter
  w0      ;      // inertia weight pada iterasi awal
  α       ;      // decrement factor
}
```

Gambar 4.3 Prosedur Inisialisasi Parameter

Seperti terlihat pada Gambar 4.3, c_1 merupakan *cognitive parameter*, c_2 merupakan *social parameter* sedangkan w^0 merupakan *inertia weight* pada iterasi awal, nilai *inertia weight* pada tiap iterasi akan mengalami penurunan akibat dari adanya α, yaitu *decrement factor*.

4.2.2 Prosedur Inisialisasi Populasi

Prosedur inisialisasi populasi digunakan untuk menginisialisasi populasi yang diperlukan sebagai solusi awal. Pengambilan populasi awal secara random dimulai dengan batas bawah 0 dan batas atas x_{max}. Penentuan x_{max} digunakan untuk membatasi agar pengambilan populasi dapat dikontrol. ρ menunjukkan banyaknya *particle* yang akan dibentuk dalam sebuah

populasi.

Secara sederhana pada algoritma PSO, populasi awal diinisialisasi seperti terlihat pada Gambar 4.4.

```
Inisialisasi Populasi
{
  Tentukan x_max ;
  for i←1 to i<=ρ
  {
     particle [i]←random[0,x_max);
  }
}
```

Gambar 4.4 Prosedur Inisialisasi Populasi

Setelah populasi awal didapatkan, berikutnya akan dibahas mengenai perubahan tiap *particle* [*i*] ke dalam bentuk *job sequence* dengan menggunakan SPV *rule*.

4.2.3 Prosedur Permutasi *Job*

Pada prosedur permutasi *job*, populasi akan diubah dari bentuk kontinu ke bentuk diskrit dengan menerapkan SPV *rule* sehingga didapatkan *job sequence* untuk tiap *particle* dan permutasi *job* untuk seluruh *particle*, seperti terlihat pada Gambar 4.5.

```
Prosedur permutasi job
{
  for i←1 to i<=ρ
  {
    Terapkan SPV rule untuk particle [i];
    Dapatkan permutasi job untuk particle [i];
  }
}
```

Gambar 4.5 Prosedur Permutasi *Job*

Penerapan SPV *rule* dijelaskan lebih lanjut pada prosedur SPV *rule* sebagai berikut. Posisi untuk *job j* pada tiap *particle* (x_{ij}) terlebih dahulu diurutkan dari nilai yang terkecil hingga nilai terbesar dengan menggunakan teknik *buble sort*. x'_{ij} merupakan posisi untuk *job j* pada tiap *particle* [i] yang

telah diurutkan. Sedangkan π_{ij} menunjukkan *job* mana yang akan dikerjakan pada urutan ke-*j* untuk *particle* [i].

Secara sederhana penerapan prosedur SPV *rule* pada permasalahan penjadwalan *permutation flowshop* dapat dilihat pada Gambar 4.6.

```
SPV rule
{
  Urutkan x[i][j]; // dari terkecil hingga terbesar
  for j←1 to j<=n // jumlah job
 {
    for k←1 to k<=n
    {
      if (x'[i][j]= x[i][k])
      {
        π[i][j]=k;v
      }
    }
  }
}
```

Gambar 4.6 Prosedur SPV *Rule*

Setelah seluruh *particle* diubah dalam bentuk *job sequence*, langkah selanjutnya adalah mendapatkan nilai *makespan* yang dilakukan dengan menerapkan prosedur evaluasi yang akan dibahas pada prosedur selanjutnya.

4.2.4 Prosedur Evaluasi

Prosedur evaluasi dilakukan setelah didapatkan *job sequence* untuk tiap *particle* [*i*] yang kemudian dihitung *makespan* dari tiap *particle* tesebut. Secara sederhana diperlihatkan pada Gambar 4.7.

```
Prosedur Evaluasi
{
  for i←1 to i<=jumlah particle
  {
    Hitung makespan tiap particle i;
    Dapatkan makespan tiap particle i;
  }
}
```

Gambar 4.7 Prosedur Evaluasi

Implementasi prosedur hitung *makespan* untuk permasalahan penjadwalan *permutation flowshop* diperlihatkan pada Gambar 4.8. Dimana *i* mewakili *job* yang akan dikerjakan sesuai dengan urutan jadwal (π). Sedangkan p(i,j) merupakan *processing time* yang disesuaikan berdasarkan jadwal permutasi yang diketahui sebelumnya.

```
Prosedur hitung makespan
{
 for i←1 to i<=n //job pada posisi i dalam permutasi job
 {
  for j←1 to j<=m // mesin j
  {
    if i=1 dan j=1)
    {
      C(i,j)=p(i,j);
    }
    else if (i ≠ 1 dan j = 1)
    {
      C(i,1)=C(i-1,1) + p(i,j);
    }
    else if (i = 1 dan j ≠ 1)
    {
      C(1,j)=C(1,j-1) + p(i,j);
    }
    else if (i ≠ 1 dan j ≠ 1)
    {
      C(i,j)= max{C(i-1,j),C(i,j-1)} + p(i,j);
    }
  }
 }
 C_max (π)= C(n,m) // makespan dari permutasi sequence π
}
```

Gambar 4.8 Hitung *Makespan*

Prosedur hitung *makespan* ini berlaku untuk semua perhitungan *makespan* yang ada pada skripsi ini. Pada pembahasan selanjutnya, *personal best* dari tiap *particle* ditentukan sesuai dengan nilai *makespan* yang terbaik selama iterasi.

4.2.5 Prosedur *Personal Best*

Pada dasarnya *personal best* digunakan untuk mencari dan menyimpan posisi terbaik yang ada pada *particle* [i] selama proses iterasi *t* berlangsung. Prosedur *personal best* diperlihatkan pada Gambar 4.9.

```
Prosedur Personal Best
{
   for t←0 to t < maksimum iterasi    // indeks untuk iterasi
   {
    if(t=0)
    {
     for i←1 to i<=ρ
     {
      Personalbest[i]=particle[i];
      Fitness[i]=C_max [i];
     }
    }
    else                              // update personal best
    {
     for i←1 to i<=ρ
     {
       if(Fitness'[i]>C_max [i])      // nilai fitness pada iterasi
                                         sebelumnya
       {
        Perbaiki fitness[i];
        Personalbest[i]=particle[i];
       }
       else
       {
         Fitness[i]=fitness'[i];
         Personalbest[i]=personalbest'[i] //personal bestpada iterasi
                                             sebelumnya
       }
     }
    }
   }
}
```

Gambar 4.9 Prosedur *Personal Best*

Berdasarkan Gambar 4.9, ilustrasi penerapan prosedur *personal best* pada permasalahan penjadwalan *permutation flowshop* adalah sebagai berikut. *Personal best* pada iterasi awal dinyatakan sebagai populasi awalnya. Hal ini berarti tiap *particle* pada populasi awal diasumsikan sebagai posisi terbaik untuk *particle* [i]. Sedangkan pada iterasi selanjutnya, *personal best* akan terus diperbaiki dari iterasi-iterasi sebelumnya.

Selain *personal best*, prosedur yang dibahas selanjutnya adalah *global best*. Prosedur ini akan menjelaskan bagaimana *personal best* yang telah didapatkan kemudian digunakan untuk mendapatkan *global best*.

4.2.6 Prosedur *Global Best*

Prosedur *global best* digunakan untuk mendapatkan posisi

terbaik dari seluruh *particle* berdasarkan pada nilai *makespan* yang paling minimum dari seluruh *particle*. Ilustrasi dari prosedur *global best* ditampilkan pada Gambar 4.10.

```
Prosedur Global Best
{
  //mencari nilai makespan terbaik dari populasi
   for i←1 to i≤ ρ
   {
    Cari  fitness[i] terkecil;      // nilai fitness personal best
   }
  //mendapatkan global best
  for t←0 to t<maksimal iterasi //iterasi yang berlangsung
  {
    if(t=0)
    {
      Fitness_global[t]=fitness [i] terkecil;
      Globalbest[t]=particle[i];
    }
    else
    {
      if(fitness_global[t-1]>fitness[i] terkecil)
      {
         Fitness_global[t]= fitness[i] terkecil;
         Perbaiki globalbest[t];
      }
      else
      {
         Fitness_global[t]=fitness_global[t-1];
         Globalbest[t]=globalbest[t-1];
      }
    }
  }

}
```

Gambar 4.10 Prosedur *Global Best*

Diawali dengan pencarian nilai terkecil dari nilai *fitness personal best*, kemudian *global best* untuk iterasi awal dinyatakan sebagai *personal best* yang dipilih berdasarkan nilai *fitness personal best* dengan nilai paling minimum. Sedangkan pada iterasi berikutnya, *global best* akan diperbaiki apabila ditemukan posisi yang jauh lebih baik.

4.2.7 Prosedur *Velocity*

Prosedur *velocity* dibagi menjadi 2 bagian yaitu, prosedur inisialisasi *velocity* dan prosedur *update velocity*. Prosedur inisialisasi *velocity* menggambarkan pembentukan *velocity* awal yang diperlihatkan pada Gambar

4.11.

```
Prosedur Inisialisasi Velocity
{
  Tentukan v_max ; // batas atas velocity yang diijinkan
  for i←1 to i<=ρ
  {
    particle [i]←random[v_min, v_max) ; // v_min = −v_max
  }
}
```

Gambar 4.11 Prosedur Inisialisasi *Velocity*

Prosedur inisialisasi *velocity* ini sama seperti prosedur inisialisasi populasi dimana diawali dengan penentuan batas atas untuk pengambilan bilangan random. Batas bawah untuk pengambilan bilangan random pada inisialisasi *velocity* merupakan bilangan negatif dari batas atas yang telah ditentukan. Sedangkan pada prosedur *update velocity* dilakukan untuk memperbaiki *velocity* yang ada sebelumnya dengan menggunakan persamaan 2.3 dan 2.1 seperti pada Gambar 4.12.

```
Prosedur Update Velocity
{
  Hitung w; //menggunakan persamaan 2.3
  for i←1 to i≤ ρ
  {
    Set velocity baru[i]; // menggunakan persamaan 2.1
  }
}
```

Gambar 4.12 Prosedur *Update Velocity*

Dari *update velocity* yang telah dilakukan kemudian didapatkan *velocity* baru yang digunakan untuk memperbaiki posisi sehingga didapatkan populasi yang baru. lebih lanjut akan dijlaskan pada prosedur *update* populasi.

4.2.8 Prosedur *Update* Populasi

Secara sederhana prosedur *update* populasi diperlihatkan pada Gambar 4.13 dimana untuk mendapatkan populasi yang baru digunakan

persamaan 2.2.

```
Prosedur Update Populasi
{
   for i←1 to i≤ ρ
   {
     Set particle baru[i]; // menggunakan persamaan 2.2
   }
}
```

Gambar 4.13 Prosedur *Update* Populasi

Posisi-posisi dalam populasi selalu diperbaiki sehingga mendekati solusi yang optimal. Selain dengan memperbaiki posisi pada tiap iterasinya, digunakan *local serch* untuk mencari posisi yang lebih baik. Pembahasan selengkapnya ditampilkan dalam prosedur *local search*.

4.2.9 Prosedur *Local Search*

Penerapan prosedur *local search* dilakukan mulai iterasi t = 1, dimana *global best* telah melalui proses *update* terlebih dahulu. Selanjutnya dari *update global best* kemudian diolah sesuai dengan prosedur *local search* dengan menggunakan VNS seperti tampak pada Gambar 4.14 untuk mendapatkan posisi yang lebih baik.

Fungsi *insert* dan *interchange* pada prosedur VNS digunakan untuk mendapatkan *job sequence* yang baru. Kedua fungsi ini bekerja secara sederhana untuk mengubah *job sequence* sebelumnya menjadi *job sequence* yang baru yaitu menyisipkan sebuah operator (*job*) yang dipilih secara acak di depan atau di belakang operator (*job*) yang dipilih secara acak pula dan mengambil dua operator (*job*) yang dipilih secara acak kemudian menukarnya untuk masing-masing fungsi. Prosedur VNS selengkapnya ditunjukkan pada Gambar 4.14 berikut.

```
Prosedur local search VNS
{
  Set π^g;
  Terapkan fungsi insert;              // dapatkan π'_baru
  Hitung makespan;
  π_baru = π'_baru;
  loop=0;
  do
  {
    kcount=0;maxmethod=2;
    do
    {
      Set π = π_baru ;
      if(kcount=0)
      {
        Terapkan fungsi insert;       // dapatkan π_baru
        Hitung makespan;
      }
      if(kcount=1)
      {
        Terapkan fungsi interchange;  // dapatkan π_baru
        Hitung makespan;
      }
      if(makespan(π'_baru)>makespan(π_baru))
      {
        π'_baru = π_baru;
        kcount=0;
      }
      else
      {
        kcount++;
      }
     While(kcount<maxmethod)
    }
   loop++;
   While(loop<n*(n-1))          // n adalah banyaknya job;
  }
if(makespan(π'_baru)≤ makespan(π^g))
{
 Perbaiki global best;         //sesuai dengan SPV rule
}

}
```

Gambar 4.14 Prosedur VNS

Global best diperbaiki setelah seluruh *looping* berakhir dengan ketentuan yang ada pada SPV *rule*. Dengan *looping* sebanyak *n**(*n*-1) diharapkan seluruh kemungkinan dari posisi yang lebih baik pada sekitar *global best* diperoleh sehingga didapatkan *global best* yang terbaik.

Berdasarkan prosedur algoritma yang telah dibahas, kemudian akan disusun program untuk penerapan *Particle Swarm Optimization* dengan *local search* untuk permasalahan penjadwalan *permutation flowshop* dengan

menggunakan bahasa pemrograman C++ Builder. Pembahasan untuk program dijelaskan pada subbab berikutnya.

4.3 Program

Program dibuat untuk memudahkan proses pencarian *job sequence* yang akan meminimalkan *makespan* dengan menggunakan bahasa pemrograman C++ Builder. Selanjutnya rincian program penerapan *Particle Swarm Optimization* dengan *local search* untuk permasalahan penjadwalan *permutation flowshop* adalah sebagai berikut :

1. **FormBegin**

   FormBegin terdiri dari 2 komponen, yaitu **opening.h** dan **opening.cpp**. **opening.h** berisi pendeklarasian data dan fungsi-fungsi yang digunakan dalam pembuatan **FormBegin** seperti mendeklarasikan daftar menu yang terdapat pada **FormBegin** yaitu menu New Problem, menu Open File, menu Exit, menu To Use, menu PSO Algorithm. **opening.cpp** bertipe C++ Builder *source* yang digunakan untuk mendefinisikan fungsi-fungsi yang telah dideklarasikan pada **opening.h**.

2. **FormInisialisasi**

   Seperti halnya pada **FormBegin**, pada **FormInisialisasi** juga terdiri dari 2 komponen utama, yaitu **inisialisasi.h** dan **inisialisasi.cpp**. **inisialisasi.h** berisi pendeklarasian data dan fungsi-fungsi yang digunakan di dalam pembuatan **FormInisialisasi**. **inisialisasi.h** juga digunakan untuk menginputkan data jumlah *job*, mesin, *processing time*, konstanta c_1, konstanta c_2, konstanta w^0 dan konstanta α. **inisialisasi.cpp** bertipe C++

Builder *source* yang digunakan untuk mendefinisikan fungsi-fungsi yang telah dideklarasikan pada **inisialisasi.h**.

3. **FormProses**

FormProses juga terdiri dari 2 komponen utama, yaitu **proses.h** dan **proses.cpp**. **proses.h** berisi pendeklarasian data dan fungsi-fungsi yang digunakan di dalam pembuatan **FormProses**. **proses.h** digunakan untuk mendapatkan solusi optimal dari permasalahan penjadwalan *permutation flowshop* berupa *job sequence*, dengan fungsi objektif meminimalkan *makespan*. **proses.cpp** bertipe C++ Builder *source* yang digunakan untuk mendefinisikan fungsi-fungsi yang telah dideklarasikan pada **proses.h**. Fungsi-fungsi yang terdapat pada **proses.h** ditunjukkan Gambar 4.15.

```
FormProses
{
  Void FormShow();                    //salinan tabel processing time dari
                                        FormInisialisasi
  Void Populasi_awal();               //menginisialisasi populasi awal
  Void Velocity_awal();               //menginisialisasi velocity awal
  Void urut1();                       //mengurutkan posisi dari populasi dari
                                        yang terkecil hingga terbesar
  Void Sequence_job();                //mengubah posisi menjadi bentuk job
                                        sequence
  Void Permutasi();                   //mendapatkan permutasi job dari
                                        seluruh particle
  Void Evaluasi();                    //melakukan evaluasi untuk seluruh
                                        particle
  Void Makespan();                    //menghitung makespan
  Void Personal_best();               //menampilkan personal best awal
  Void Global_best();                 //menampilkan global best awal
  Void Inertia_weight();              //menampilkan inertia weight
  Void Update_velocity(int t)();      //melakukan update velocity
  Void Update_populasi();             //melakukan update populasi
  Void Update_personalbest();         //melakukan update personal best
  Void Update_globalbest();           //melakukan update global best
  Void Local_search();                //penerapan local search
}
```

Gambar 4.15 proses.h

Fungsi-fungsi yang terdapat pada Gambar 4.15 merupakan fungsi-fungsi yang digunakan untuk menjalankan algoritma PSO dengan *local search*. Pada subbab

berikutnya akan dibahas mengenai data yang digunakan untuk diselesaikan secara manual maupun dengan program C++ Builder.

4.4 Data

Terdapat dua data contoh permasalahan penjadwalan *permutation flowshop* digunakan pada skripsi ini, sebagai berikut :

1. Data I

Data I adalah permasalahan 4-*job* 3-mesin yang diambil dari Ignall dan Schrage (1964) dan terdapat pada Lampiran 1.1. Dari data tersebut kemudian akan dicari *job sequence* dengan nilai *makespan* yang minimal. Data I akan diselesaikan secara manual serta menggunakan program C++ Builder.

2. Data II

Data II adalah permasalahan 20-*job* 5-mesin dan permasalahan 20-*job* 10-mesin yang merupakan data dari Taillard (1989) yang terdapat pada Lampiran 1.2 dan Lampiran 1.3. Program yang telah dibuat diimplementasikan pada Data II.

Kedua data tersebut merupakan salah satu contoh dari kasus berukuran kecil dan kasus berukuran besar. Hal ini digunakan untuk melihat kinerja dari algoritma PSO dengan *local search* dalam menyelesaikan masing-masing kasus untuk ukuran permasalahan yang berbeda. Penyelesaian untuk contoh kasus dengan menggunakan data I secara manual dibahas pada subbab selanjutnya.

4.5 Contoh Kasus Permasalahan Penjadwalan *Permutation Flowshop* dengan Menggunakan Data 4-*Job* 3-Mesin yang Diselesaikan Secara Manual

Permasalahan yang akan diselesaikan yaitu 4 *job* yang dikerjakan pada 3 mesin untuk tiap *job*. *Processing time* pada tiap mesin disajikan pada Lampiran

1.1. Maksimal iterasi yang dikerjakan sebanyak 1 iterasi. Adapun langkah-langkah yang digunakan dalam menyelesaikan permasalahan penjadwalan *permutation flowshop* menggunakan algoritma PSO dengan *local search* sebagai berikut :

- t = 0, merupakan iterasi yang berisi inisialisasi-inisialisasi yang digunakan untuk diproses pada iterasi t = 1. Inisialisasi yang dilakukan terlebih dahulu adalah inisialisasi parameter, dilanjutkan dengan melakukan inisialisasi populasi untuk mendapatkan populasi awal serta melakukan inisialisasi *velocity* untuk mendapatkan *velocity* awal. Sedangkan *personal best* dan *global best* pada iterasi t = 0 didapatkan setelah populasi awal diperoleh.

4.5.1 Inisialisasi Parameter

Pada penyelesaian permasalahan ini, ditentukan nilai-nilai parameter yang akan digunakan dalam algoritma PSO. Parameter – parameter tersebut adalah c_1= c_2= 2, w^0= 0,9, α= 0,95. Pemilihan nilai parameter berdasarkan pada Tasgetiren dkk (2004). Inisialisasi parameter digunakan untuk memperbaiki *velocity* pada iterasi berikutnya.

4.5.2 Inisialisasi Populasi

Inisialisasi populasi dilakukan dengan mengambil sejumlah bilangan random yang merepresentasikan kandidat-kandidat solusi untuk permasalahan penjadwalan *permutation flowshop*. Menurut Tasgetiren dkk (2004), banyaknya *particle* (ρ) sebuah populasi yang dibentuk sebanyak 2 x jumlah *job* (n) dengan batas atas (x_{max}) yang dipilih adalah 4, populasi awal yang dapat dibangkitkan dengan bilangan random [0, 4) diperlihatkan pada

Tabel 4.1.

Tabel 4.1 Populasi Awal

Particle	***Position Value***			
	x_{i1}	x_{i2}	x_{i3}	x_{i4}
X_1	1,253	0,416	2,833	1,472
X_2	0,925	3,416	1,703	2,689
X_3	3,321	1,226	0,213	0,758
X_4	2,264	0,837	0,519	3,247
X_5	1,775	2,564	1,436	0,603
X_6	3,223	3,809	1,791	2,917
X_7	0,524	1,3	2,673	0,942
X_8	2,963	0,869	1,471	2,868

4.5.3 Inisialisasi *Velocity*

Seperti pada pembentukan populasi awal, hal yang sama dilakukan pada pembentukan *velocity* awal, yaitu dengan membangkitkan sejumlah bilangan random yang digunakan untuk pergerakan *particle* pada iterasi selanjutnya. Menurut Tasgetiren dkk (2004), batas atas (v_{max}) yang dipilih untuk membentuk *velocity* awal adalah 4 dan batas bawah (v_{min}) adalah $-v_{max}$ selanjutnya *velocity* awal yang dapat dibentuk dengan membangkitkan bilangan random [-4, 4) diperlihatkan pada Tabel 4.2.

Tabel 4.2 *Velocity* Awal

Velocity	***Velocity Value***			
	v_{i1}	v_{i2}	v_{i3}	v_{i4}
V_1	2,495	−1,675	0,234	−0,936
V_2	1,243	3,221	−2,465	−1,157
V_3	0,386	−1,883	2,281	−2,473
V_4	−3,208	2,846	−0,937	1,805
V_5	0,114	−3,017	1,906	2,036
V_6	−1,783	0,281	−2,112	0,164

Velocity	v_{i1}	v_{i2}	v_{i3}	v_{i4}
V_7	2,897	−0,366	1,478	3,741
V_8	2,481	3,042	−3,817	−0,488

4.5.4 Permutasi *Job*

Langkah selanjutnya setelah didapatkan populasi awal adalah mendapatkan permutasi *job* untuk seluruh *particle*. Tiap *particle i* pada populasi awal akan diubah ke dalam bentuk *job sequence* dengan menerapkan SPV *rule*. Implementasi SPV *rule* pada seluruh *particle* akan ditunjukkan oleh *particle* 1 pada Tabel 4.3.

Tabel 4.3 Implementasi SPV *Rule* Untuk *Particle* 1

Dimensi (*j*)	**1**	**2**	**3**	**4**
x_{1j}^0	1.253	0.416	2.883	1.472
x'^0_{1j}	0.416	1.253	1.472	2.883
π_{1j}^0	2	1	4	3

Dari Tabel 4.3 didapatkan *job sequence* untuk *particle* 1 adalah 2-1-4-3. Dengan menggunakan cara yang sama untuk i = 2, ..., ρ, permutasi *job* pada tiap *particle* dapat dilihat pada Tabel 4.4.

Tabel 4.4 Permutasi *Job* Pada Iterasi 0

***Particle* (*i*)**	**Permutasi *Job* (π_i^0)**
1	2-1-4-3
2	1-3-4-2
3	3-4-2-1
4	3-2-1-4
5	4-3-1-2
6	3-4-1-2
7	1-4-2-3
8	2-3-4-1

4.5.5 Evaluasi

Untuk langkah evaluasi, *job sequence* yang didapat dari langkah sebelumnya kemudian dievaluasi dengan menggunakan kriteria *makespan*. Pada kriteria ini, total waktu yang dibutuhkan tiap mesin untuk mengerjakan *job-job* yang dijadwalkan akan dihitung dengan menggunakan persamaan-persamaan berikut:

$$\left.\begin{aligned} C(\pi_1, 1) &= p_{\pi_1,1} \\ C(\pi_j, 1) &= C(\pi_{j-1}, 1) + p_{\pi_j,1} \quad ; j = 2, \ldots, n \\ C(\pi_1, k) &= C(\pi_1, k-1) + p_{\pi_1,k} \quad ; k = 2, \ldots, m \\ C(\pi_j, k) &= \max\{C(\pi_{j-1}, k), C(\pi_j, k-1)\} + p_{\pi_j,k} \; ; \; j = 2, \ldots, n \; ; \; k = 2, \ldots, m \end{aligned}\right\} \quad (4.1)$$

Selanjutnya perhitungan *makespan* dari *job sequence particle* 1 dilakukan dengan beberapa langkah berikut.

- Buat tabel *processing time job sequence particle* 1 seperti pada Tabel 4.5.

Tabel 4.5 *Processing Time* Untuk π_1^0

	M1	**M2**	**M3**
***Job* 2**	7	12	16
***Job* 1**	13	3	12
***Job* 4**	2	6	1
***Job* 3**	26	9	7

- Kemudian hitung *completion time* untuk seluruh *job* yang dikerjakan pada masing-masing mesin *k* dengan menggunakan persamaan (4.1), seperti berikut.
  - Untuk $j = 1$
    - $k = 1$

      $C(\pi_1, 1) = p_{\pi_1,1}$

      $= 7$

- k = 2
  $C(\pi_1, 2) = C(\pi_1, 1) + p_{\pi_1,2}$
  = 7 +12
  = 19
- k = 3
  $C(\pi_1, 3) = C(\pi_1, 2) + p_{\pi_1,3}$
  = 19 + 16
  = 35

- Untuk j = 2
  - k = 1
    $C(\pi_2, 1) = C(\pi_1, 1) + p_{\pi_2,1}$
    = 7 + 13
    = 20
  - k = 2
    $C(\pi_2, 2) = \max\{C(\pi_1, 2), C(\pi_2, 1)\} + p_{\pi_2,2}$
    = max {20, 19} +3
    = 20 + 3
    = 23
  - k = 3
    $C(\pi_2, 3) = \max\{C(\pi_1, 3), C(\pi_2, 2)\} + p_{\pi_2,3}$
    = max {35, 23} +12
    = 35 +12
    = 47

- Dengan menggunakan cara yang serupa untuk j = 3, 4 dan k = 1, 2, 3, *completion time* untuk seluruh *job* pada masing-masing mesin diperlihatkan pada Tabel 4.6.

Tabel 4.6 Perhitungan *Completion Time* Seluruh *Job*

	$C(\pi_j, 1)$	$C(\pi_j, 2)$	$C(\pi_j, 3)$
$C(\pi_1, k)$	7	19	35
$C(\pi_2, k)$	20	23	47
$C(\pi_3, k)$	22	29	48
$C(\pi_4, k)$	48	57	**64**

Berdasarkan Tabel 4.6, *makespan* yang didapatkan untuk *particle* 1 adalah 64. Selanjutnya untuk π_i^0 dengan i = 2, 3, ..., ρ dilakukan langkah yang serupa untuk mendapatkan *makespan*. Hasil *makespan* untuk seluruh *particle* kemudian disajikan dalam Tabel 4.7

Tabel 4.7 Hasil *Makespan* Pada Iterasi 0

***Particle* (*i*)**	**Permutasi *Job* (π_i^0)**	***Makespan*** **(f_i^0)**
1	2-1-4-3	64
2	1-3-4-2	82
3	3-4-2-1	81
4	3-2-1-4	76
5	4-3-1-2	76
6	3-4-1-2	76
7	1-4-2-3	64
8	2-3-4-1	63

Berdasarkan *makespan* yang didapat untuk masing-masing *particle* kemudian akan digunakan untuk mendapatkan *personal best* seperti pada langkah selanjutnya.

4.5.6 *Personal Best*

Personal best digunakan untuk menyimpan posisi terbaik yang memilki nilai *makespan* minimum pada tiap *particle* nya. Pada iterasi awal, populasi awal diasumsikan berisi posisi-posisi terbaik dengan nilai *makespan* tiap *particle* digunakan sebagai nilai *fitness* untuk *personal best* (f_i^p), sehingga *personal best* untuk iterasi awal adalah sebagai berikut.

Tabel 4.8 *Personal Best* Pada Iterasi 0

i	***Personal Best***				f_i^p
	p_{i1}	p_{i2}	p_{i3}	p_{i4}	
1	1,253	0,416	2,833	1,472	64
2	0,925	3,416	1,703	2,689	82
3	3,321	1,226	0,213	0,758	81
4	2,264	0,837	0,519	3,247	76
5	1,775	2,564	1,436	0,603	76
6	3,223	3,809	1,791	2,917	76
7	0,524	1,3	2,673	0,942	64
8	2,963	0,869	1,471	2,868	63

4.5.7 *Global Best*

Global best digunakan untuk menyimpan posisi terbaik dari *swarm*. Pada langkah ini, nilai *makespan* yang paling minimum dari nilai *fitness pesonal best* digunakan sebagai nilai *fitness global best*. Perolehan nilai *makespan* yang paling minimum dari seluruh nilai *fitness personal best* diperlihatkan pada Tabel 4.9.

Tabel 4.9 Pencarian f_l^0

***Particle* (*i*)**	f_i^p	$f_l^0 = \min\{f_i^p\}$
1	64	$f_l^0 = f_8^p = 63$
2	82	
3	81	
4	76	
5	76	
6	76	
7	64	
8	**63**	

Berdasarkan Tabel 4.9, *particle* 8 mempunyai nilai *makespan* (f_8^0) yang paling minimum dibandingkan dengan nilai *makespan* pada *particle* lain yaitu 63. Sehingga *global best* pada iterasi awal adalah *particle* 8, diperlihatkan pada Tabel 4.10.

Tabel 4.10 *Global Best* Pada Iterasi 0

Global Best				f^g
g_1	g_2	g_3	g_4	
2,963	0,869	1,471	2,868	63

- *t* = 1, merupakan iterasi yang berisi proses algoritma PSO dengan *local search*. Proses yang terjadi diawali dengan *update velocity* yang dibentuk dari

velocity awal, *personal best* dan *global best* sebelumnya. Dilanjutkan dengan *update* populasi setelah itu diubah dalam bentuk *job sequence* dengan proses permutasi *job* kemudian dilakukan evaluasi sehingga *update personal best* dan *global best* didapatkan. Proses terakhir adalah menerapkan *local search*.

4.5.8 *Update Velocity*

Pada tiap iterasi, *velocity* akan selalu diperbaiki dengan menggunakan persamaan (2.1), langkah-langkah yang digunakan untuk mendapatkan *velocity* yang baru adalah sebagai berikut.

- Pilih r_1 dan r_2 dengan membangkitkan secara random [0, 1], misalnya r_1 = 0,179 dan r_2 = 0,363.
- Kemudian tentukan *velocity* dari *particle i* dengan menggunakan persamaan (2.1) seperti berikut.
  - Untuk *i* = 1
    - *j* = 1

      v_{11}^1 = 0,9*2,495 + 2*0,179*(1,253 – 1,253) + 2*0,363*(2,963–1,253)

      = 4,96899

      Karena $v_{11}^1 > v_{max}$ maka $v_{11}^1 = 4$.
    - *j* = 2

      v_{12}^1 = 0,9* (–1,675) + 2*0,179*(0,416 – 0,416) + 2*0,363*(0,869 – 0,416)

      = –0,78602
    - *j* = 3

      v_{13}^1 = 0,9* 0,234 + 2*0,179*(2,833 – 2,833) + 2*0,363*(1,471 – 2,833)

      = –1,95863
    - *j* = 4

      v_{14}^1 = 0,9* (–0,936) + 2*0,179*(1,472 – 1,472) + 2*0,363*(2,868 – 1,472)

      = 1,380984
  - Untuk *i* = 2, 3, ..., ρ gunakan cara sejalan dengan *i* = 1 sehingga diperoleh *velocity* yang baru untuk setiap *particle* seperti pada

Tabel 4.11.

Tabel 4.11 *Velocity* Pada Iterasi 1

Velocity	***Velocity Value***			
	v_{i1}	v_{i2}	v_{i3}	v_{i4}
V_1	4	−0,78602	−1,95863	1,380984
V_2	4	−1,15766	−2,588	−0,75621
V_3	−0,22278	−2,26329	4	1,134859
V_4	−1,77392	2,612366	0,672933	1,020874
V_5	1,994706	−4	1,771144	4
V_6	−2,0188	−4	−2,41046	0,069559
V_7	4	−1,01585	−0,5842	4
V_8	−2,2329	2,7378	−3,4353	−0,4392

Seperti yang terlihat pada Tabel 4.11, *velocity value* berada pada interval [-4, 4], hal ini berarti posisi dapat bergerak dengan *velocity* maksimal sebesar 4 dan *velocity* minimal sebesar -4. *Velocity* yang baru kemudian akan digunakan untuk memperbaiki populasi.

4.5.9 *Update* Populasi

Seperti halnya *velocity*, populasi pada tiap iterasi juga diperbaiki. Populasi yang baru didapatkan dengan menggunakan persamaan (2.2), perhitungan untuk mendapatkan populasi baru adalah sebagai berikut.

- Untuk *i* = 1
  - *j* = 1
    x_{11}^1 = 1,253 + 4
    = 5,253
  - *j* = 2
    x_{12}^1 = 0,416 + (–0,78602)
    = –0,37002
  - *j* = 3
    x_{13}^1 = 2,833 + (–1,95863)
    = 0,874367
  - *j* = 4

x_{14}^{1} = 1,472 + 1,380984
= 2,852984

- Untuk i = 2, 3, ..., ρ gunakan cara sejalan dengan i = 1 sehingga diperoleh populasi yang baru diperlihatkan pada Tabel 4.12.

Tabel 4.12 Populasi Pada Iterasi 1

Particle	***Position Value***			
	x_{i1}	x_{i2}	x_{i3}	x_{i4}
X_1	5,253	−0,37002	0,874367	2,852984
X_2	4,925	2,258339	−0,885	1,93279
X_3	3,09822	−1,03729	4,213	1,892859
X_4	0,490085	3,449366	1,191933	4,267874
X_5	3,769706	−1,436	3,207144	4,603
X_6	1,204203	−0,191	−0,61946	2,986559
X_7	4,524	0,284154	2,88796	4,942
X_8	0,7301	3,6068	−1,9643	2,4288

4.5.10 Permutasi *Job*

Sama seperti pada iterasi sebelumnya, tiap *particle* pada populasi akan diubah ke dalam bentuk *job sequence* dengan menerapkan SPV *rule*. Permutasi *job* untuk seluruh *particle* pada iterasi 1 dapat dilihat pada Tabel 4.13.

Tabel 4.13 Permutasi *Job* Pada Iterasi 1

***Particle* (*i*)**	**Permutasi *Job* (π_i^1)**
1	2-3-4-1
2	3-4-2-1
3	2-4-1-3
4	1-3-2-4
5	2-3-1-4
6	3-2-1-4
7	2-3-1-4
8	3-1-4-2

4.5.11 Evaluasi

Dengan menggunakan langkah yang sama seperti pada iterasi sebelumnya dalam mengevaluasi tiap *particle* dengan kriteria *makespan*, *makespan* yang didapatkan untuk seluruh *particle* ditampilkan dalam Tabel 4.14.

Tabel 4.14 Hasil *Makespan* Pada Iterasi 1

Particle **(*i*)**	**Permutasi *Job*** **(π_i^1)**	***Makespan*** **(f_i^1)**
1	2-3-4-1	63
2	3-4-2-1	81
3	2-4-1-3	64
4	1-3-2-4	77
5	2-3-1-4	62
6	3-2-1-4	76
7	2-3-1-4	62
8	3-1-4-2	76

4.5.12 *Update Personal Best*

Pada langkah *update personal best*, nilai *fitness personal best* pada iterasi sebelumnya akan dibandingkan dengan nilai *makespan* pada iterasi sekarang yang kemudian digunakan untuk memperbaiki *personal best* pada tiap *particle*. Tabel 4.15 menampilkan perubahan nilai *fitness personal best*.

Tabel 4.15 *Update* f_i^p

***Particle* (*i*)**	f_i^p	f_i^1	f_i^p **baru**
1	64	**63**	**63**
2	82	**81**	**81**
3	81	**64**	**64**
4	**76**	77	**76**
5	76	**62**	**62**
6	**76**	76	**76**
7	64	**62**	**62**
8	**63**	76	**63**

Tabel 4.15 menunjukkan pembentukan nilai *fitness personal best* (f_i^p) yang baru. Selanjutnya *personal best* diperbaiki dengan melihat nilai *fitness personal best* nya.

Tabel 4.16 *Personal Best* Pada Iterasi 1

I	***Personal Best***				f_i^p
	p_{i1}	p_{i2}	p_{i3}	p_{i4}	
1	5,253	−0,37002	0,874367	2,852984	63
2	4,925	2,258339	−0,885	1,93279	81
3	3,09822	−1,03729	4,213	1,892859	64
4	2,264	0,837	0,519	3,247	76
5	3,769706	−1,436	3,207144	4,603	62
6	3,223	3,809	1,791	2,917	76
7	4,524	0,284154	2,88796	4,942	62
8	2,963	0,869	1,471	2,868	63

4.5.13 *Update Global Best*

Seperti pencarian *global best* pada iterasi t = 0, terlebih dahulu dilakukan pencarian terhadap nilai *makespan* yang paling minimum dari seluruh nilai *fitness personal best* seperti tampak pada Tabel 4.17.

Tabel 4.17 Pencarian f_l^1

***Particle* (*i*)**	f_i^p	$f_l^1 = \min\{f_i^p\}$
1	63	$f_l^1 = f_5^p = 62$
2	81	
3	64	
4	76	
5	**62**	
6	76	
7	62	
8	63	

Berdasarkan Tabel 4.17, nilai *makespan* yang paling minimum terdapat pada *particle* 5 dan *particle* 7 yaitu 62. Karena keduanya memiliki nilai yang paling minimum maka dipilih salah satu dari kedua *particle* tersebut yaitu *particle* 5. Kemudian nilai *fitness global best* (f^g) pada iterasi sebelumnya dibandingkan dengan f_l^1. Karena $f_l^1 < f^g$ maka nilai *fitness global best* diperbaiki sehingga *global best* yang diperoleh dari *particle* 5 diperlihatkan pada Tabel 4.18.

Tabel 4.18 *Global Best* Pada Iterasi 1

Global Best				f^g
g_1	g_2	g_3	g_4	
3,769706	−1,436	3,207144	4,603	62

4.5.14 *Local Search*

Local search yang digunakan adalah *Variable Neighborhood Search* (VNS). Penerapan VNS adalah sebagai berikut.

1. Bentuk s_0 yang merupakan *job sequence* dari π^g sehingga s_0 adalah 2-3-

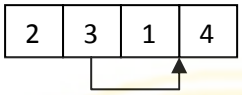
1-4.

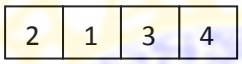
2. Nilai η, κ dipilih secara random dari *integer* [1, 4], misalkan η = 2, κ = 4.

   Terapkan operasi *insert* pada permutasi *job* s_0 dengan operator $\eta = 2$,

   $\kappa = 4$

   $s = insert\ (s_0, \eta, \kappa)$

   $s_0 =$ 

   $s =$ 

   Sehingga permutasi *job s* adalah 2-1-3-4.

3. Kemudian hitung *makespan* dari permutasi *job s* dengan menggunakan cara yang serupa seperti pada langkah evaluasi. Perhitungan *completion time* untuk mendapatkan nilai *makespan* dari permutasi *job s* diperlihatkan pada Tabel 4.19.

Tabel 4.19 Perhitungan *Completion Time* Permutasi *Job s*

	$C(\pi_j, 1)$	$C(\pi_j, 2)$	$C(\pi_j, 3)$
$C(\pi_1, k)$	7	19	35
$C(\pi_2, k)$	20	23	47
$C(\pi_3, k)$	46	55	62
$C(\pi_4, k)$	48	61	**63**

   Berdasarkan Tabel 4.19 maka nilai *makespan* yang didapat untuk permutasi *job s* adalah 63.

4. Atur loop = 0, kcount = 0, kemudian lakukan operasi *neighborhood* sesuai dengan prosedur VNS yang terdapat pada Gambar 4.14.

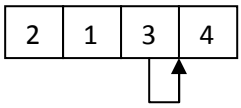
   Untuk loop = 0

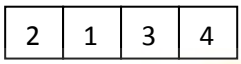
   - kcount = 0

Misal η = 3 dan κ = 4

s_1 = *insert* (s, η, κ)

= *insert* $(s, 3, 4)$

s = 

s_1 = 

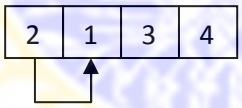
Sehingga permutasi *job* s_1 adalah 2-1-3-4. Karena permutasi *job* s_1 sama seperti permutasi *job* s, maka nilai *makespan* untuk permutasi *job* $s_1(f(s_1))$ adalah 63 dan nilai kcount bertambah 1.

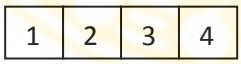
- kcount =1

  Misal η = 1 dan κ = 2

  s_1 = *interchange* (s, η, κ)

  = *interchange* $(s, 1, 2)$

  s = 

  s_1 = 

  Sehingga permutasi *job* s_1 adalah 1-2-3-4, kemudian cari nilai *makespan*nya dengan menggunakan perhitungan *completion time* seperti pada Tabel 4.20.

Tabel 4.20 Perhitungan *Completion Time* Permutasi *Job* s_1

	$C(\pi_j, 1)$	$C(\pi_j, 2)$	$C(\pi_j, 3)$
$C(\pi_1, k)$	13	16	28
$C(\pi_2, k)$	20	32	48
$C(\pi_3, k)$	46	55	62
$C(\pi_4, k)$	48	61	**63**

Berdasarkan Tabel 4.20 maka nilai *makespan* yang didapat untuk permutasi *job* s_1 adalah 63. Karena nilai *makespan* dari permutasi *job* s_1 sama dengan nilai *makespan* dari permutasi *job s* maka dilanjutkan pada *looping* berikutnya.

5. Lakukan hal yang sama untuk loop = 1, 3, .., *n**(*n*-1)-1. Hasil dari seluruh *looping* disederhanakan dalam Tabel 2.21.

Tabel 4.21 Implementasi VNS

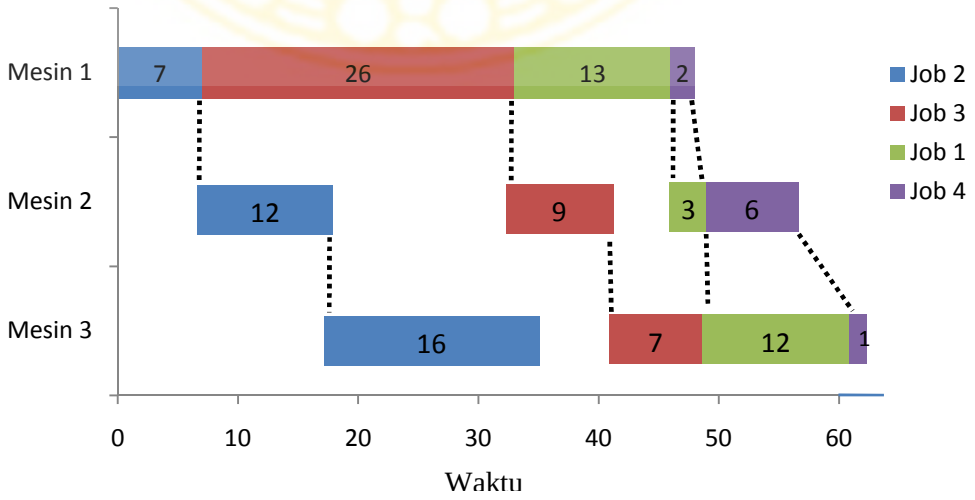
Iterasi	Operasi	η	κ	*s*	*f(s)*	s_1	$f(s_1)$
loop=0	*Insert*	3	4	2 1 3 4	63	2 1 3 4	63
	Interchange	1	2	2 1 3 4	63	1 2 3 4	63
loop=1	*Insert*	4	3	2 1 3 4	63	2 1 3 4	63
	Interchange	4	1	2 1 3 4	63	4 1 3 2	78
loop=2	*Insert*	1	2	2 1 3 4	63	2 1 3 4	63
	Interchange	3	4	2 1 3 4	63	2 1 4 3	64
loop=3	*Insert*	1	4	2 1 3 4	63	1 3 2 4	77
	Interchange	2	1	2 1 3 4	63	1 2 3 4	63
loop=4	*Insert*	3	1	2 1 3 4	63	2 3 1 4	62
	Insert	4	2	2 3 1 4	62	2 3 4 1	63
	Interchange	1	3	2 3 1 4	62	1 3 2 4	77
loop=5	*Insert*	4	2	2 3 1 4	62	2 3 4 1	63
	Interchange	2	3	2 3 1 4	62	2 1 3 4	63
loop=6	*Insert*	3	2	2 3 1 4	62	2 3 1 4	62
	Interchange	2	4	2 3 1 4	62	2 4 1 3	64
loop=7	*Insert*	1	3	2 3 1 4	62	3 2 1 4	76
	Interchange	4	1	2 3 1 4	62	4 3 1 2	76
loop=8	*Insert*	3	4	2 3 1 4	62	2 3 1 4	62
	Interchange	2	1	2 3 1 4	62	3 2 1 4	76
loop=9	*Insert*	3	1	2 3 1 4	62	2 1 3 4	63
	Interchange	2	3	2 3 1 4	62	2 1 3 4	63
loop=10	*Insert*	4	1	2 3 1 4	62	2 4 3 1	63
	Interchange	2	4	2 3 1 4	62	2 4 1 3	64
loop=11	*Insert*	1	4	2 3 1 4	62	3 1 2 4	75
	Interchange	3	4	2 3 1 4	62	2 3 4 1	63

Setelah dilakukan *looping* sebanyak *n**(*n*-1) kemudian akan dibandingkan nilai *makespan* dari permutasi *job s* pada loop terakhir dengan nilai *fitness global best*. Karena nilai *makespan* keduanya bernilai sama dengan *job sequence* yang sama pula maka π^g = 2-3-1-4 dengan nilai *fitness global best* adalah 62.

Jadwal yang diperoleh kemudian akan ditampilkan dalam bentuk *gantt chart* untuk mempermudah melihat proses dari tiap *job* yang dikerjakan pada masing-masing mesin. Pembahasan selebihnya mengenai *gantt chart* ditampilkan pada subbab berikutnya.

4.5.15 *Gantt Chart*

Berdasarkan perhitungan yang dilakukan sebelumnya untuk permasalahan penjadwalan *permutation flowshop* pada 4-*job* 3-mesin sebanyak 2 iterasi, didapatkan jadwal yang suboptimal yaitu 2-3-1-4 dengan nilai *makespan* yaitu 62. Selanjutnya dari hasil yang diperoleh akan digambarkan dalam bentuk *gantt chart* seperti pada Gambar 4.16.



Gambar 4.16 *Gantt Chart* Untuk Jadwal 2-3-1-4

Cara membaca *gantt chart* di atas adalah sebagai berikut :

a. *Job* 2 diproses pada mesin 1 mulai dari waktu ke-0 dengan *processing time* sebesar 7 satuan waktu sehingga proses selesai pada waktu ke-7, dilanjutkan pada mesin 2 dengan waktu mulai proses yaitu waktu ke-7 hingga waktu ke-19 dengan *processing time* sebesar 12 satuan waktu. Setelah *job* 2 diproses pada mesin 2 kemudian dilanjutkan pada mesin 3 dengan *processing time* sebesar 16 satuan waktu sehingga *job* 2 selesai dikerjakan pada mesin 3 pada waktu ke-35. Cara membaca tersebut juga berlaku untuk *job* 3, *job* 1, dan *job* 4.

b. Apabila *gantt chart* dibaca secara horizontal, maka pada mesin 2 terdapat rentang waktu antara *job* 2 dan *job* 3 sebesar 14 satuan waktu. Hal ini disebabkan *job* 2 telah selesai dikerjakan pada waktu ke-19 untuk mesin 2, sedangkan *job* 3 baru selesai dikerjakan pada waktu ke-33 untuk mesin 1. Begitu juga penyebab terjadinya rentang waktu antara *job-job* lainnya pada suatu mesin *j*.

c. Sedangkan apabila *gantt chart* dibaca secara vertikal, maka terdapat garis putus-putus yang menghubungkan antara proses *job i* pada mesin *j* dengan proses *job i* pada mesin $j + 1$. Bila diperhatikan tidak semua garis membentuk garis vertikal, tetapi ada yang membentuk sudut. Misalnya pada *job* 4 yang dikerjakan pada mesin 1 kemudian dilanjutkan pengerjaan pada mesin 2. Hal ini menunjukkan adanya *idle time* atau waktu tunggu bagi *job* 4 untuk diproses pada mesin 2. Proses *job* 4 pada mesin 1 telah selesai pada waktu ke-48 sedangkan proses *job* 1 pada

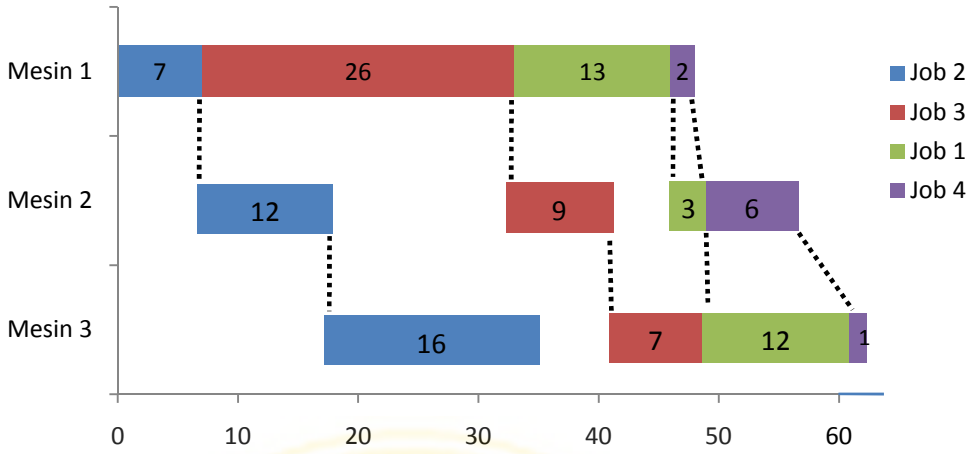
mesin 2 baru selesai pada waktu ke-49 sehingga *job* 4 baru bisa diproses oleh mesin 2 pada waktu ke-49. Jadi *idle time* untuk *job* 4 adalah selama 1 satuan waktu. Penjelasan tersebut juga berlaku bagi semua *job i*.

4.6 Implementasi Program Pada Contoh Kasus Permasalahan Penjadwalan *Permutation Flowshop*

Program yang telah dibuat untuk permasalahan penjadwalan *permutation flowshop* dengan menggunakan algoritma PSO dapat diterapkan dalam contoh kasus untuk 4-*job* 3-mesin dimana *processing time* untuk tiap *job* terdapat pada Lampiran 1.1. Parameter-parameter yang digunakan sebagai input awal adalah :

- Jumlah *particle* = 20
- x_{max} = 4
- v_{max} = 4
- Jumlah iterasi = 10
- c_1 = 2
- c_2 = 2
- w^0 = 0,9
- α = 0,95

Solusi yang didapat untuk permasalahan 4-*job* 3-mesin adalah jadwal 2-3-1-4 dengan nilai *makespan* sebesar 62 satuan waktu. Jadwal yang telah diperoleh kemudian ditampilkan dalam bentuk *gantt chart* seperti terlihat pada Gambar 4.17. Sedangkan rincian hasil implementasi program untuk permasalahan penjadwalan 4-*job* 3-mesin dapat dilihat pada Lampiran 3.



Gambar 4.17 *Gantt Chart* Untuk Solusi Permasalahan 4-*Job* 3-Mesin

4.7 Perbandingan Hasil Perhitungan Dengan Parameter Yang Berbeda Menggunakan PSO Dengan *Local Search*

Pada perbandingan ini, data yang digunakan adalah permasalahan penjadwalan *permutation flowshop* untuk 20-*job* 5-mesin dan 2-*job* 10-mesin yang dapat dilihat pada Lampiran 1.2 dan Lampiran 1.3. Berikut hasil perhitungan *makespan* yang didapat pada masing-masing permasalahan dengan parameter yang berbeda, yaitu nilai w^0 dan α. Pada perbandingan ini menggunakan jumlah *particle* = 20, jumlah iterasi = 10, c_1 = 2, c_2 = 2, x_{max} = 4, v_{max} = 4.

1) Data 20-*Job* 5-Mesin

Dengan menggunakan program yang telah dibuat kemudian dihitung nilai *makespan* dengan nilai w^0 dan α yang berbeda-beda untuk data 20-*job* 5-mesin. Hasil perhitungan *makespan* diperlihatkan pada Tabel 4.22.

Tabel 4.22 *Makespan* untuk Data 20-*Job* 5-Mesin

Nilai *makespan*		α								
		0,1	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**
w^0	**0,1**	1297	1278	1297	1297	1297	1297	1297	1297	1297
	0,2	1297	1297	1297	1297	1297	1293	1297	1297	1297
	0,3	1297	1297	1297	1297	1279	1278	1297	1297	1297
	0,4	1297	1278	1297	1297	1297	1297	1283	1297	1297
	0,5	1297	1297	1297	1297	1283	1297	1294	1297	1297
	0,6	1297	1278	1297	1297	1297	1297	1297	1297	1297
	0,7	1297	1297	1297	1297	1297	1297	1297	1297	1297
	0,8	1297	1297	1297	1297	1297	1297	1297	1297	1297
	0,9	1278	1297	1297	1297	1297	1297	1278	1297	1278

Berdasarkan dari Tabel 4.22, terlihat bahwa *makespan* dengan nilai terkecil sebesar 1278 didapat untuk w^0 = 0,1, α = 0,2, w^0 = 0,3, α = 0,6, w^0 = 0,4, α = 0,2, w^0 = 0,6, α = 0,2, w^0 = 0,9, α = 0,1, w^0 = 0,9, α = 0,7 dan w^0 = 0,9, α = 0,9. Sedangkan sebagian besar *makespan* bernilai 1297, hal ini menunjukkan adanya kekonvegenan solusi.

2) Data 20-*Job* 10-Mesin

Hasil *makespan* untuk permasalahan 20-*job* 10-mesin dimana nilai parameter w^0 dan α yang digunakan berbeda-beda, terlihat pada Tabel 4.23.

Tabel 4.23 *Makespan* Untuk Data 20-*Job* 10-Mesin

Nilai *makespan*		α								
		0,1	**0,2**	**0,3**	**0,4**	**0,5**	**0,6**	**0,7**	**0,8**	**0,9**
w^0	**0,1**	1601	1602	1594	1615	1586	1603	1595	1598	1597
	0,2	1593	1595	1594	1600	1604	1602	1586	1593	1611
	0,3	1595	1595	1592	1604	1592	1591	1603	1603	1596
	0,4	1596	1605	1595	1596	1599	1605	1593	1593	1594
	0,5	1599	1595	1596	1592	1593	1593	1586	1593	1597
	0,6	1587	1592	1588	1609	1598	1595	1594	1596	1593
	0,7	1602	1587	1594	1591	1598	1592	1598	1593	1595
	0,8	1598	1596	1596	1593	1598	1592	1594	1601	1598
	0,9	1594	1605	1586	1600	1603	1587	1597	1597	1589

Berdasarkan Tabel 4.23 tampak bahwa *makespan* dengan nilai terkecil sebesar 1586 didapat untuk w^0 = 0,1 dan α = 0,5, w^0 = 0,2 dan α = 0,7, w^0 = 0,5 dan α = 0,7, w^0 = 0,9 dan α = 0,3. Sedangkan w^0 dan α dengan kombinasi yang lain memiliki nilai *makespan* yang berbeda-beda, hal ini menunjukkan tidak ada hubungan antara nilai w^0 yang membesar dan α mengecil dengan nilai *makespan* atau nilai w^0 yang mengecil dan α membesar dengan nilai *makespan*.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

1. Implementasi program untuk contoh kasus menggunakan data 4-*job* 3-mesin dengan jumlah *particle* = 20, maksimum iterasi = 10, x_{max} = 4, v_{max} = 4, c_1 = 2, c_2 = 2, w^0 = 0,9, α = 0,95 diperoleh solusi yaitu jadwal 2-3-1-4 dengan *makespan* sebesar 62 satuan waktu.
2. Dengan menggunakan nilai parameter w^0 dan α yang berbeda-beda, didapatkan hasil *makespan* yaitu, untuk data 20-*job* 5-mesin memiliki nilai *makespan* yang terkecil sebesar 1278 dengan w^0 = 0,1, α = 0,2, w^0 = 0,3, α = 0,6, w^0 = 0,4, α = 0,2, w^0 = 0,6, α = 0,2, w^0 = 0,9, α = 0,1, w^0 = 0,9, α = 0,7 dan w^0 = 0,9, α = 0,9. Sedangkan untuk data 20-*job* 10-mesin didapatkan nilai *makespan* yang terkecil sebesar 1586 dengan w^0 = 0,1 dan α = 0,5, w^0 = 0,2 dan α = 0,7, w^0 = 0,5 dan α = 0,7, w^0 = 0,9 dan α = 0,3.

5.2 Saran

Pada permasalahan penjadwalan *permutation flowshop* menggunakan algoritma *Particle Swarm Optimization* dengan *local search* untuk data dengan ukuran yang cukup besar sering terjadi minimum lokal secara dini sehingga pada penelitian selanjutnya disarankan untuk menggunakan *Hybrid Particle Swarm*

Optimization dengan *Simulated Annealing*, diharapkan dengan menambahkan algoritma *Simulated Annealing* minimum lokal tidak akan terjadi sehingga solusi yang optimal bisa ditemukan.

DAFTAR PUSTAKA

Basu, R., 2008, *Implementing Six Sigma and Lean : A Practical Guide to Tools and Techniques*, Linacre House, Jordan Hill, Oxford OX2 8DP, UK.

Bennatan, E.M., 1995, *On Time, within Budget: Software Project Management Practice and Techniques*, 2nd Edition, Subject Industrial Project Management- Computer Programs, New York.

Gould, F.E., 1997, *Managing The Construction Process: Estimating, Scheduling, and Project Control*, Upper Saddle River, N.J.

Hassan, R., 2004, *Particle Swarm Optimization : Method and Applications*, Engineering System Division, Massachusetts Institude of Technology.

Heryanto, I. dan Raharjo, B., 2006, *Pemrograman Borland C++ Builder*, Informatika, Bandung.

Hoos, H.H dan Stuzle, T., 2005, *Stochastic Local Search : Foundations and Applications*, Morgan Kufmann.

Ignall, E. dan Schrage, L., 1964, Application of The Branch and Bound Technique To Some Flow-Shop Scheduling Problem, *Operations Research*, Vol. 13, No. 3, pp. 400-412.

Kennedy, J. dan Eberhart, R., 1995, Particle Swarm Optimization, *Neural Networks*, Proceedings, IEEE International Conference on, Vol 4, pp. 1942-1948.

Mladenovic, N. dan Hansen, P., 1997, Variable Neighborhood Search, *Computers Operation Research*, Vol. 24, No. 11, pp. 1097-1100.

Morton, T.E. dan Pentico, D.W., 1993, *Heuristic Scheduling System: With Applications to Production System and Project Management*, John Wiley Series in Engineering & Technology Management, Canada, pp. 295-323.

Pinedo, M., 2002, *Scheduling Theory, Algorithm, and System*, Second Edition, New York University.

Poli, R., Kennedy, J., dan Blackwell, T., 2007, Particle Swarm Optimization, *Swarm Intell*, Springer, 1: 33-57.

Soetanto, T.V. dan Soetanto, D.P., 1999, Penjadwalan Flowshop dengan Algoritma Genetika, *Jurnal Teknik Industri*, Vol. 1, No. 1, pp. 1-11.

Taillard, E., 1989, *Benchmarks For Basic Scheduling Problem*, ORWP89/21.

Tasgetiren, M.F., Sevkli, M., Liang, Y.-C., dan Gencyilmaz, G., 2004, Particle Swarm Optimization Algorithm for Permutation Flowshop Sequencing Problem, *M. Dorigo et al. (Eds.): ANTS,* LNCS 3172, pp. 382-389.

Teugeh, M., Soeprijanto, dan Purnomo, H.M., 2009, Modified Improved Particle Swarm Optimization for Optimal Generator Scheduling, *Seminar Nasional Aplikasi Teknologi Informasi,* Yogyakarta, ISSN: 1907-5022.

Ucar, H. dan Tasgetiren, M.F., A Particle Swarm Optimization Algorithm for Permutation Flow Shop Sequencing Problem with The Number of Tardy Jobs Criterion, website: www.*iseresearch.eng. wayne.edu/2006/Proceedings2006/Hatice*, diakses tanggal 16 Juni 2009.

Uysal, O. dan Bulkan, S., 2008, Comparison of Genetic Algorithm and Particle Swarm Optimization for Bicriteria Permutation Flowshop Scheduling Problem, *International Journal of Computational Intelligence Research,* ISSN: 0973-1873, Vol. 4, No. 2, pp. 159-175.

Lampiran 1 : Data *Processing Time* Untuk Permasalahan *Permutation Flowshop*

1.1 Tabel Data *Processing Time* Untuk 4-*Job* 3-Mesin

	M1	**M2**	**M3**
***Job* 1**	13	3	12
***Job* 2**	7	12	16
***Job* 3**	26	9	7
***Job* 4**	2	6	1

Sumber : Ignall, E. dan Schrage, L., 1964, *Application of The Branch and Bound Technique To Some Flow-Shop Scheduling Problem*, Cornell University, New York, pp. 400-412.

1.2 Tabel Data *Processing Time* Untuk 20-*Job* 5-Mesin

	M1	M2	M3	M4	M5
***Job* 1**	54	79	16	66	58
***Job* 2**	83	3	89	58	56
***Job* 3**	15	11	49	31	20
***Job* 4**	71	99	15	68	85
***Job* 5**	77	56	89	78	53
***Job* 6**	36	70	45	91	35
***Job* 7**	53	99	60	13	53
***Job* 8**	38	60	23	59	41
***Job* 9**	27	5	57	49	69
***Job* 10**	87	56	64	85	13
***Job* 11**	76	3	7	85	86
***Job* 12**	91	61	1	9	72
***Job* 13**	14	73	63	39	8
***Job* 14**	29	75	41	41	49
***Job* 15**	12	47	63	56	47
***Job* 16**	77	14	47	40	87
***Job* 17**	32	21	26	54	58
***Job* 18**	87	86	75	77	18
***Job* 19**	68	5	77	51	68
***Job* 20**	94	77	40	31	28

Sumber : Taillard, E., 1989, *Benchmarks For Basic Scheduling Problem*, ORWP89/21.

1.3 Tabel Data *Processing Time* Untuk 20-*Job* 10-Mesin

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
***Job* 1**	74	28	89	60	54	92	9	4	25	15
***Job* 2**	21	3	52	88	66	11	8	18	15	84
***Job* 3**	58	27	56	26	12	54	88	25	91	8
***Job* 4**	4	61	13	58	57	97	72	28	49	30
***Job* 5**	21	34	7	76	70	57	27	95	56	95
***Job* 6**	28	76	32	98	82	53	22	51	10	79
***Job* 7**	58	64	32	29	99	65	50	84	62	9
***Job* 8**	83	87	98	47	84	77	2	18	70	91
***Job* 9**	31	54	46	79	16	51	49	6	76	76
***Job* 10**	61	98	60	26	41	36	82	90	99	26
***Job* 11**	94	76	23	19	23	53	93	69	58	42
***Job* 12**	44	41	87	48	11	19	96	61	83	66
***Job* 13**	97	70	7	95	68	54	43	57	84	70
***Job* 14**	94	43	36	78	58	86	13	5	64	91
***Job* 15**	66	42	26	77	30	40	60	75	74	67
***Job* 16**	6	79	85	90	5	56	11	4	14	3
***Job* 17**	37	88	7	24	5	79	37	38	18	98
***Job* 18**	22	15	34	10	39	74	91	28	48	4
***Job* 19**	99	49	36	85	58	24	84	4	96	71
***Job* 20**	83	72	48	55	31	3	67	80	86	62

Sumber : Taillard, E., 1989, *Benchmarks For Basic Scheduling Problem*, ORWP89/21.

Lampiran 2 : *Source Code* Program

opening.h

```
//---------------------------------------------------------------------------

#ifndef openingH
#define openingH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <jpeg.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
class TFormBegin : public TForm
{
__published:   // IDE-managed Components
        TMainMenu *MainMenu1;
        TImage *Image1;
        TLabel *Label1;
        TLabel *Label2;
        TMenuItem *File1;
        TMenuItem *Help1;
        TMenuItem *About1;
        TMenuItem *NewProblem1;
        TMenuItem *OpenFile1;
        TMenuItem *ToUse1;
        TMenuItem *PSOAlgorithm1;
        TLabel *Label3;
        TMenuItem *OpenData1;
        TMenuItem *Exit1;
        void __fastcall NewProblem1Click(TObject *Sender);
        void __fastcall OpenFile1Click(TObject *Sender);
        void __fastcall Exit1Click(TObject *Sender);
        void __fastcall OpenData1Click(TObject *Sender);

private:       // User declarations
public:        // User declarations
        __fastcall TFormBegin(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFormBegin *FormBegin;
//---------------------------------------------------------------------------
#endif
```

opening.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "opening.h"
#include "inisialisasi.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormBegin *FormBegin;
//---------------------------------------------------------------------------
__fastcall TFormBegin::TFormBegin(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormBegin::NewProblem1Click(TObject *Sender)
```

```
{
  if(FormInisialisasi != NULL)
        {
            delete FormInisialisasi;
        }
        FormInisialisasi = new TFormInisialisasi(this);
        FormInisialisasi->Caption="Form New";
        FormInisialisasi->Button_next1->Visible=true;
        FormInisialisasi->Button_ok1->Visible=true;
        //FormInisialisasi->Job->Enabled=true;
        //FormInisialisasi->Mesin->Enabled=true;
        FormInisialisasi->Button_browse->Visible=false;
        FormInisialisasi->Button_data->Visible=false;
        FormInisialisasi->Show();
}
//---------------------------------------------------------------------------

void __fastcall TFormBegin::OpenFile1Click(TObject *Sender)
{
  if(FormInisialisasi != NULL)
        {
            delete FormInisialisasi;
        }
        FormInisialisasi = new TFormInisialisasi(this);
        FormInisialisasi->Caption="Form Open";
        FormInisialisasi->Button_next1->Visible=true;
        FormInisialisasi->Button_ok1->Visible=false;
        FormInisialisasi->Button_browse->Enabled=true;
        FormInisialisasi->Button_data->Visible=false;
        FormInisialisasi->Show();
}
//---------------------------------------------------------------------------
void __fastcall TFormBegin::Exit1Click(TObject *Sender)
{
  try
     {
        Close();
     }
     catch(EInvalidPointer &E)
     {
        Close();
     }

}
//---------------------------------------------------------------------------

void __fastcall TFormBegin::OpenData1Click(TObject *Sender)
{
  if(FormInisialisasi != NULL)
        {
            delete FormInisialisasi;
        }
        FormInisialisasi = new TFormInisialisasi(this);
        FormInisialisasi->Caption="Form Open";
        FormInisialisasi->Button_next1->Visible=true;
        FormInisialisasi->Button_ok1->Visible=false;
        FormInisialisasi->Button_browse->Visible=false;
        FormInisialisasi->Button_data->Enabled=true;
        FormInisialisasi->Show();
}
//---------------------------------------------------------------------------
```

inisialisasi.h

```
//---------------------------------------------------------------------------

#ifndef inisialisasiH
#define inisialisasiH
//---------------------------------------------------------------------------
```

```
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <Grids.hpp>
#include <jpeg.hpp>
//---------------------------------------------------------------------------
class TFormInisialisasi : public TForm
{
__published:   // IDE-managed Components
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *Edit_mesin;
        TEdit *Edit_job;
        TButton *Button_ok1;
        TGroupBox *GB_proctime;
        TEdit *Edit_c1;
        TEdit *Edit_c2;
        TEdit *Edit_w;
        TEdit *Edit_alpha;
        TEdit *Edit_maxiter;
        TEdit *Edit_pop;
        TEdit *Edit_xmax;
        TEdit *Edit_vmax;
        TButton *Button_next1;
        TButton *Button_back1;
        TButton *Button_browse;
        TOpenDialog *OD_browse;
        TLabel *Label11;
        TCheckBox *CB_c1;
        TCheckBox *CB_c2;
        TCheckBox *CB_w;
        TCheckBox *CB_alpha;
        TStringGrid *SG_proctime;
        TImage *Image1;
        TButton *Button_data;
        TOpenDialog *OD_data;
        void __fastcall Button_browseClick(TObject *Sender);
        void __fastcall Inisialisasi_Variabel(TObject *Sender);
        void __fastcall Button_ok1Click(TObject *Sender);
        void __fastcall Edit_c1Change(TObject *Sender);
        void __fastcall Edit_c2Change(TObject *Sender);
        void __fastcall Edit_wChange(TObject *Sender);
        void __fastcall Edit_alphaChange(TObject *Sender);
        void __fastcall Button_next1Click(TObject *Sender);
        void __fastcall Button_back1Click(TObject *Sender);
        void __fastcall Button_dataClick(TObject *Sender);

private:       // User declarations
public:        // User declarations
        __fastcall TFormInisialisasi(TComponent* Owner);
        int mesin,job,maxiter,pop,i,j,k;
        float c1,c2,w0,alpha,xmax,vmax,vmin;
};
//---------------------------------------------------------------------------
extern PACKAGE TFormInisialisasi *FormInisialisasi;
//---------------------------------------------------------------------------
#endif
```

inisialisasi.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <iostream.h>

#include "inisialisasi.h"
#include "opening.h"
#include "proses.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormInisialisasi *FormInisialisasi;
//---------------------------------------------------------------------------
__fastcall TFormInisialisasi::TFormInisialisasi(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Inisialisasi_Variabel(TObject *Sender)
{
  mesin=StrToInt(Edit_mesin->Text);
  job=StrToInt(Edit_job->Text);
  c1=StrToFloat(Edit_c1->Text);
  c2=StrToFloat(Edit_c2->Text);
  w0=StrToFloat(Edit_w->Text);
  alpha=StrToFloat(Edit_alpha->Text);
  maxiter=StrToInt(Edit_maxiter->Text);
  pop=StrToInt(Edit_pop->Text);
  xmax=StrToFloat(Edit_xmax->Text);
  vmax=StrToFloat(Edit_vmax->Text);
}
//---------------------------------------------------------------------------
void __fastcall TFormInisialisasi::Button_ok1Click(TObject *Sender)
{
   mesin=StrToInt(Edit_mesin->Text);
   job=StrToInt(Edit_job->Text);
   SG_proctime->ColCount=mesin+1;
   SG_proctime->RowCount=job+1;

   for(i=1;i<=job;i++) //baris
   {

    for(j=1;j<=mesin;j++)   //kolom
    {
      SG_proctime->Cells[j][0]="Mesin "+ String(j);
      SG_proctime->Cells[0][i]="Job "+ String(i);
    }
   }
}
//---------------------------------------------------------------------------
void __fastcall TFormInisialisasi::Button_browseClick(TObject *Sender)
{
  if(OD_browse->Execute())
     {
        std::auto_ptr<TStrings> LoadStrings(new TStringList());
        LoadStrings->LoadFromFile(OD_browse->FileName);
        job = StrToInt(LoadStrings->Strings[0]);
        mesin= StrToInt(LoadStrings->Strings[1]);
        c1=StrToFloat(LoadStrings->Strings[2]);
        c2=StrToFloat(LoadStrings->Strings[3]);
        w0=StrToFloat(LoadStrings->Strings[4]);
        alpha=StrToFloat(LoadStrings->Strings[5]);
        Edit_job->Text = job;
        Edit_mesin->Text = mesin;
```

```
        Edit_c1->Text=c1;
        Edit_c2->Text=c2;
        Edit_w->Text=w0;
        Edit_alpha->Text=alpha;
        SG_proctime->ColCount = mesin+1;
        SG_proctime->RowCount = job+1;
        for(j=1;j<=mesin;j++)
        {
            SG_proctime->Cells[j][0]=" Mesin "+String(j);
        }
        k=6;
        for(i=1;i<=job;i++)
        {
            SG_proctime->Cells[0][i]=" Job "+String(i);
        }
        for(i=1;i<SG_proctime->RowCount;++i)
        {
           for(j=1;j<SG_proctime->ColCount;++j)
           {
              SG_proctime->Cells[j][i]=LoadStrings->Strings[k++];
           }
        }
     }
}
//---------------------------------------------------------------------------
void __fastcall TFormInisialisasi::Edit_c1Change(TObject *Sender)
{
try
     {
      c1=StrToFloat(Edit_c1->Text);
      if(c1 > 2.0)
      {
       CB_c1->Checked = false;
       CB_c1->Caption = "SALAH!! 1.5 <= C1 <= 2.0";
      }
      else if (c1 < 1.5)
      {
       CB_c1->Checked = false;
       CB_c1->Caption = "SALAH!! 1.5 <= C1 <= 2.0";
      }
      else if (c1>=1.5 && c1<=2.0)
      {
       CB_c1->Checked = true;
       CB_c1->Caption = "BENAR";
      }
     }
     catch (EConvertError &E)
     {
      CB_c1->Checked = false;
      CB_c1->Caption = "Input C1";
     }
}
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Edit_c2Change(TObject *Sender)
{
try
     {
      c2=StrToFloat(Edit_c2->Text);
      if(c2 > 2.5)
      {
       CB_c2->Checked = false;
       CB_c2->Caption = "SALAH!! 2.0 <= C1 <= 2.5";
      }
      else if (c2 < 2.0)
      {
       CB_c2->Checked = false;
       CB_c2->Caption = "SALAH!! 2.0 <= C1 <= 2.5";
      }
      else if (c2>=2.0 && c2<=2.5)
      {
```

```
        CB_c2->Checked = true;
        CB_c2->Caption = "BENAR";
       }
      }
      catch (EConvertError &E)
      {
       CB_c2->Checked = false;
       CB_c2->Caption = "Input C2";
      }

}
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Edit_wChange(TObject *Sender)
{
      try
      {
       w0=StrToFloat(Edit_w->Text);
       if(w0 >= 1.0)
       {
        CB_w->Checked = false;
        CB_w->Caption = "SALAH!! 0 < W0 < 1";
       }
       else if (w0 <= 0)
       {
        CB_w->Checked = false;
        CB_w->Caption = "SALAH!! 0 < W0 < 1";
       }
       else if (w0>0 && w0<1)
       {
        CB_w->Checked = true;
        CB_w->Caption = "BENAR";
       }
      }
      catch (EConvertError &E)
      {
       CB_w->Checked = false;
       CB_w->Caption = "Input W0";
      }

}
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Edit_alphaChange(TObject *Sender)
{
      try
      {
       alpha=StrToFloat(Edit_alpha->Text);
       if(alpha >= 1.0)
       {
        CB_alpha->Checked = false;
        CB_alpha->Caption = "SALAH!! 0 < Alpha < 1";
       }
       else if (alpha <= 0)
       {
        CB_alpha->Checked = false;
        CB_alpha->Caption = "SALAH!! 0 < Alpha < 1";
       }
       else if (alpha>0 && alpha<1)
       {
        CB_alpha->Checked = true;
        CB_alpha->Caption = "BENAR";
       }
      }
      catch (EConvertError &E)
      {
       CB_alpha->Checked = false;
       CB_alpha->Caption = "Input Alpha";
      }

}
```

```
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Button_next1Click(TObject *Sender)
{
try
   {
    if(FormProses != NULL)
    {
      FormInisialisasi->WindowState = wsMinimized;
      FormInisialisasi->WindowState = wsNormal;
      FormProses->Show();
      FormProses->WindowState = wsMaximized;
    }
    else
    {
      FormInisialisasi->WindowState = wsMinimized;
      FormProses = new TFormProses(this);
      FormProses->Show();
      FormProses->WindowState = wsMaximized;
    }
  }
  catch (Exception &exception)
  {
    FormInisialisasi->WindowState = wsMinimized;
    FormProses = new TFormProses(this);
    FormProses->Show();
    FormProses->WindowState = wsMaximized;
  }
}
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Button_back1Click(TObject *Sender)
{
    if(FormBegin != NULL)
    {
        FormBegin->Show();
        FormBegin->WindowState = wsMaximized;
        FormInisialisasi->WindowState = wsMinimized;
    }
}
//---------------------------------------------------------------------------

void __fastcall TFormInisialisasi::Button_dataClick(TObject *Sender)
{
  if(OD_data->Execute())
     {
        std::auto_ptr<TStrings> LoadStrings(new TStringList());
        LoadStrings->LoadFromFile(OD_data->FileName);
        job = StrToInt(LoadStrings->Strings[0]);
        mesin= StrToInt(LoadStrings->Strings[1]);
        Edit_job->Text = job;
        Edit_mesin->Text = mesin;
        SG_proctime->ColCount = mesin+1;
        SG_proctime->RowCount = job+1;
        for(j=1;j<=mesin;j++)
        {
            SG_proctime->Cells[j][0]=" Mesin "+String(j);
        }
        k=2;
        for(i=1;i<=job;i++)
        {
            SG_proctime->Cells[0][i]=" Job "+String(i);
        }
        for(i=1;i<SG_proctime->RowCount;++i)
        {
           for(j=1;j<SG_proctime->ColCount;++j)
           {
              SG_proctime->Cells[j][i]=LoadStrings->Strings[k++];
           }
        }
     }
```

```
}
//---------------------------------------------------------------------------
```

proses.h

```
//---------------------------------------------------------------------------

#ifndef prosesH
#define prosesH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Grids.hpp>
#include <ComCtrls.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TFormProses : public TForm
{
__published:   // IDE-managed Components
        TStringGrid *SG_copy1;
        TStringGrid *SG_populasi;
        TStringGrid *SG_velocity;
        TStringGrid *SG_urut1;
        TStringGrid *SG_job;
        TStringGrid *SG_start;
        TStringGrid *SG_fglobal;
        TStringGrid *SG_urut2;
        TStringGrid *SG_fitness;
        TStringGrid *SG_end;
        TStringGrid *SG_plama;
        TStringGrid *SG_fp;
        TStringGrid *SG_popbaru;
        TStringGrid *SG_velbaru;
        TStringGrid *SG_jobtime;
        TStringGrid *SG_w;
        TButton *Button_proses1;
        TStringGrid *SG_fp1;
        TStringGrid *SG_personal;
        TStringGrid *SG_global;
        TStringGrid *SG_iterasijob;
        TStringGrid *SG_iterasiposisi;
        TStringGrid *SG_iterasimakespan;
        TStringGrid *SG_local;
        TStringGrid *SG_nilailocal;
        TStringGrid *SG_simpan;
        TStringGrid *SG_pilih;
        TStringGrid *SG_hasil;
        TStringGrid *SG_poplama;
        TStringGrid *SG_vellama;
        TStringGrid *SG_globaljob;
        TStringGrid *SG_urut2lama;
        TStringGrid *SG_joblama;
        TGroupBox *GroupBox1;
        TRichEdit *RE_hasil;
        TBitBtn *BitBtn_proses;
        TEdit *Edit1;
        TEdit *Edit2;
        TEdit *Edit3;
        TEdit *Edit4;
        TRichEdit *RE_proses;
        TEdit *Edit5;
        TEdit *Edit6;
        TEdit *Edit7;
        TEdit *Edit8;
        TEdit *Edit9;
        TEdit *Edit10;
        TEdit *Edit11;
        TEdit *Edit12;
```

```
        TEdit *Edit13;
        TEdit *Edit14;
        TEdit *Edit15;
        TEdit *Edit16;
        TEdit *Edit17;
        TEdit *Edit18;
        TEdit *Edit19;
        TEdit *Edit20;
        TEdit *Edit21;
        TEdit *Edit22;
        TEdit *Edit23;
        TEdit *Edit24;
        TEdit *Edit25;
        TEdit *Edit26;
        TEdit *Edit27;
        TEdit *Edit28;
        TEdit *Edit29;
        TEdit *Edit30;
        TEdit *Edit31;
        TBitBtn *BitBtn_print;
        TBitBtn *BitBtn_save;
        TBitBtn *BitBtn_ganttchart;
        void __fastcall FormShow(TObject *Sender);
        void __fastcall Populasi_awal();
        void __fastcall Velocity_awal();
        void __fastcall urut1();
        void __fastcall Sequence_job();
        void __fastcall Permutasi();
        void __fastcall Evaluasi();
        void __fastcall Makespan();
        void __fastcall urut2();
        void __fastcall Ambil_random();
        void __fastcall Insert();
        void __fastcall Interchange();
        void __fastcall Evaluasi1();
        void __fastcall Local_search();
        void __fastcall Perbaiki();
        void __fastcall PSO();
        void __fastcall Mean();
        void __fastcall Standart_deviasi();
        void __fastcall Personal_Best(int t);
        void __fastcall Global_Best(int t);
        void __fastcall Inertia_Weight();
        void __fastcall Update_Velocity(int t);
        void __fastcall Simpan();
        void __fastcall Update_Populasi();
        void __fastcall Simpan1();
        void __fastcall Hasil(int t);
        void __fastcall BitBtn_prosesClick(TObject *Sender);
        void __fastcall BitBtn_printClick(TObject *Sender);
        void __fastcall BitBtn_saveClick(TObject *Sender);
        void __fastcall BitBtn_ganttchartClick(TObject *Sender);


private:       // User declarations
public:        // User declarations
        __fastcall TFormProses(TComponent* Owner);
        int i,j,pop,mesin,job,proctime,thp,k,ii,a,b,c,nilai,x,y,x1,data1,data2;
        int data,z,maxiter,t,pilih,aa,ab,ac,ad,pilih1,tetha,tetha1,kappa,kappa1;
        int selisih,simpan,selisih1,simpan1,jj,kk,loop,kcount,urutan,l,s,makespan;
        //int jummakespan;
        int
baris,kolom,makespan1,makespan2,f,save,banyak,g,jumlahbaris,h,particle;
        int jumjob;
        float xmax,bil,n,vmax,vmin,m,tmp,w0,alpha,inertia,c1,c2,r,rr,r1,r2;
        float w,velocity,personal,posisi,global,velocity1,tmp1,banding,banding1;
        float mean,xi,sigma,SD,iterasi,jummakespan;
        float acak,acak1,velocitybaru,posisi1,velocitylama;
};
//---------------------------------------------------------------------------
```

```
extern PACKAGE TFormProses *FormProses;
//---------------------------------------------------------------------------
#endif
```

proses.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>

#include "opening.h"
#include "inisialisasi.h"
#include "proses.h"
#include "print1.h"
#include "print2.h"
#include "save1.h"
#include "save2.h"
#include "gantt_chart.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormProses *FormProses;
//---------------------------------------------------------------------------
__fastcall TFormProses::TFormProses(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::FormShow(TObject *Sender)
{
   pop=StrToInt(FormInisialisasi->pop);
   job=StrToInt(FormInisialisasi->job);
   mesin=StrToInt(FormInisialisasi->mesin);
   SG_copy1->ColCount=mesin+1;
   SG_copy1->RowCount=job+1;
   for(i=0;i<SG_copy1->RowCount;i++)
   {
     for(j=0;j<SG_copy1->ColCount;j++)
     {
       if(i==0 && j!=0)
       {
         SG_copy1->Cells[j][i]=j;
       }
       if(j==0 && i!=0)
       {
         SG_copy1->Cells[j][i]=i;
       }
       if(i!=0 && j!=0)
       {
         proctime=StrToInt(FormInisialisasi->SG_proctime->Cells[j][i]);
         SG_copy1->Cells[j][i]=proctime;
       }
     }
   }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Populasi_awal()
{
  job=StrToInt(FormInisialisasi->Edit_job->Text);
  pop=StrToInt(FormInisialisasi->Edit_pop->Text);
  SG_populasi->ColCount=job+1;
  SG_populasi->RowCount=pop+1;
  xmax=StrToFloat(FormInisialisasi->Edit_xmax->Text);
```

```
  RE_proses->Lines->Add("\n========== Populasi Awal ==========");
  for(i=0;i<SG_populasi->RowCount;i++)
  {
   for(j=0;j<SG_populasi->ColCount;j++)
   {
    if(i!=0 && j==0)
     {
       SG_populasi->Cells[j][i]="Particle "+ String(i);
     }
     else if (i==0 && j!=0)
     {
       SG_populasi->Cells[j][i]="Job "+ String(j);
     }
     else if(i!=0 && j!=0)
     {
       bil=random(10000);
          if(bil!=0)
          {
            n=(xmax*bil)/10000;
            SG_populasi->Cells[j][i]=n;
          }
     }
   }
  }

}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Velocity_awal()
{
  job=StrToInt(FormInisialisasi->Edit_job->Text);
  pop=StrToInt(FormInisialisasi->Edit_pop->Text);
  SG_velocity->ColCount=job+1;
  SG_velocity->RowCount=pop+1;
  vmax=StrToFloat(FormInisialisasi->Edit_vmax->Text);
  vmin=-vmax;
  RE_proses->Lines->Add("\n========== Velocity Awal ==========");
  for(i=0;i<SG_velocity->RowCount;i++)
  {
   for(j=0;j<SG_velocity->ColCount;j++)
   {
     if(i!=0 && j==0)
     {
       SG_velocity->Cells[j][i]="Particle "+ String(i);
     }
     else if (i==0 && j!=0)
     {
       SG_velocity->Cells[j][i]="Job "+ String(j);
     }
     else if(i!=0 && j!=0)
     {
       bil=random(10000);
          if(bil!=0)
          {
            m=vmin+((2*vmax*bil)/10000);
            SG_velocity->Cells[j][i]=m;
          }
     }
   }
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::urut1()
{
    pop=StrToInt(FormInisialisasi->Edit_pop->Text);
    job=StrToInt(FormInisialisasi->Edit_job->Text);
    SG_urut1->RowCount=pop+1;
    SG_urut1->ColCount=job+1;
    for(i=0;i<SG_urut1->RowCount;i++)
    {
      for(j=0;j<SG_urut1->ColCount;j++)
      {
```

```
          SG_urut1->Cells[j][i]=SG_populasi->Cells[j][i];
        }
      }
      //teknik bubble sort
      for(i=1;i<=pop;i++)    //baris
      {
       for(j=1;j<=job-1;j++)
       {
        for(k=j+1;k<=job;k++)
        {
          banding=StrToFloat(SG_urut1->Cells[j][i]);
          banding1=StrToFloat(SG_urut1->Cells[k][i]);
          if(banding>banding1)
          {
           tmp=StrToFloat(SG_urut1->Cells[j][i]);
           SG_urut1->Cells[j][i]=SG_urut1->Cells[k][i];
           SG_urut1->Cells[k][i]=tmp;
          }
        }
       }
      }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Sequence_job()
{
   //ShowMessage("permutasi job");
   pop=StrToInt(FormInisialisasi->Edit_pop->Text);
   job=StrToInt(FormInisialisasi->Edit_job->Text);
   SG_job->ColCount=job+1;
   SG_job->RowCount=pop+1;
   SG_pilih->ColCount=job;
   SG_pilih->RowCount=2;

   for(i=1;i<=SG_job->RowCount-1;i++)
    {
      for(j=1;j<=SG_job->ColCount-1;j++)
      {
         SG_job->Cells[0][i]="Particle "+ String(i);
         SG_job->Cells[j][0]="Job ke-"+ String(j);
      }
    }

    for(a=0;a<job;a++)
    {
     SG_pilih->Cells[a][0]=a+1;
    }
    for(b=1;b<=pop;b++)   //baris
    {
       for(i=0;i<job;i++) //kolom pada SG_pilih
       {
         SG_pilih->Cells[i][1]=SG_populasi->Cells[i+1][b];
       }

       for(j=1;j<=job;j++)
       {
         for(k=0;k<=job-j;k++)
         {
          if(StrToFloat(SG_urut1->Cells[j][b])==StrToFloat(SG_pilih->Cells[k][1]))
           {
             for(l=0;l<2;l++)
             {
               tmp=StrToFloat(SG_pilih->Cells[k][l]);
                for(s=k;s<=job-j-1;s++)
                {
                  SG_pilih->Cells[s][l]=SG_pilih->Cells[s+1][l];
                }
               SG_pilih->Cells[job-j][l]=tmp;
            }
          }
          SG_job->Cells[j][b]=SG_pilih->Cells[job-j][0];
        }
```

```
      }
    }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Permutasi()
{
 RE_proses->Lines->Add("\n==========    Permutasi    Job    ==========\n    ---Dengan
Menggunakan SPV Rule---");
 urut1();
 Sequence_job();

}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Makespan()
{
  mesin=StrToInt(FormInisialisasi->Edit_mesin->Text);
  job=StrToInt(FormInisialisasi->Edit_job->Text);
  SG_start->ColCount=mesin+1;
  SG_start->RowCount=job+1;
  SG_end->ColCount=mesin+1;
  SG_end->RowCount=job+1;
   for(i=1;i<SG_start->RowCount;i++)
   {
    for(j=1;j<SG_start->ColCount;j++)
    {
     SG_start->Cells[0][i]=SG_jobtime->Cells[0][i];
     SG_start->Cells[j][0]=SG_jobtime->Cells[j][0];
     SG_end->Cells[0][i]=SG_jobtime->Cells[0][i];
     SG_end->Cells[j][0]=SG_jobtime->Cells[j][0];
    }
   }
   for(i=1;i<=job;i++) //baris
   {
     for(j=1;j<=mesin;j++) //kolom
     {
       if(i==1 && j==1)
       {
         SG_start->Cells[j][i]=0;
         SG_end->Cells[j][i]=SG_jobtime->Cells[j][i];
       }
       else if(i==1 && j>1)
       {
         SG_start->Cells[j][i]=SG_end->Cells[j-1][i];
       }
       else if(i>1 && j==1)
       {
         SG_start->Cells[j][i]=SG_end->Cells[j][i-1];
       }
       else if(i>1 && j>1)
       {
         if(StrToInt(SG_end->Cells[j-1][i])>StrToInt(SG_end->Cells[j][i-1]))
         {
           SG_start->Cells[j][i]=SG_end->Cells[j-1][i];
         }
         else
         {
           SG_start->Cells[j][i]=SG_end->Cells[j][i-1];
         }
       }
       SG_end->Cells[j][i]=StrToInt(SG_start->Cells[j][i])+StrToInt(SG_jobtime-
>Cells[j][i]);
     }
   }
    nilai=StrToInt(SG_end->Cells[SG_end->ColCount-1][SG_end->RowCount-1]);
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Evaluasi()
{
   mesin=StrToInt(FormInisialisasi->Edit_mesin->Text);
   job=StrToInt(FormInisialisasi->Edit_job->Text);
   pop=StrToInt(FormInisialisasi->Edit_pop->Text);
```

```
   SG_jobtime->ColCount=mesin+1;
   SG_jobtime->RowCount=job+1;
   SG_fitness->ColCount=2;
   SG_fitness->RowCount=pop+1;
   RE_proses->Lines->Add("\n ========== Evaluasi ==========");
   for(ii=1;ii<=pop;ii++)
    {
     for(a=1;a<=job;a++)  //kolom pada SG_job
     {
       for(b=1;b<=job;b++) //baris pada SG_copy1
       {
         if(StrToInt(SG_job->Cells[a][ii])==b)
         {
           for(c=1;c<=mesin;c++)
           {
             SG_jobtime->Cells[0][a]=SG_copy1->Cells[0][b];
             SG_jobtime->Cells[c][0]="mesin "+ String(c);
             SG_jobtime->Cells[c][a]=SG_copy1->Cells[c][b];
           }
         }
       }
     }
      Makespan();
      SG_fitness->Cells[1][ii]=nilai;
      SG_fitness->Cells[0][ii]=ii;
      SG_fitness->Cells[1][0]="Makespan";
   }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Personal_Best(int t)
{
 SG_personal->ColCount=job+1;
 SG_personal->RowCount=pop+1;
 SG_plama->ColCount=job+1;
 SG_plama->RowCount=pop+1;
 SG_fp->ColCount=2;
 SG_fp->RowCount=pop+1;
 SG_fp1->ColCount=2;
 SG_fp1->RowCount=pop+1;
 RE_proses->Lines->Add("\n========== Personal Best ==========");
 for(i=0;i<pop+1;i++)      //baris
 {
  for(j=0;j<job+1;j++)     //kolom
  {
   if(i!=0 && j==0)
   {
     SG_personal->Cells[j][i]="Particle "+String(i);
   }
   else if(i==0 && j!=0)
   {
     SG_personal->Cells[j][i]="Job "+String(j);
   }
  }
 }

 if(t==0)
 {
  for(a=1;a<pop+1;a++)
  {
   Edit11->Text=" ";
   for(b=1;b<job+1;b++)
   {
    SG_personal->Cells[b][a]=SG_populasi->Cells[b][a];
    Edit11->Text=Edit11->Text+"  "+String(SG_personal->Cells[b][a]);
   }
   RE_proses->Lines->Add("Particle "+String(a)+"\n"+Edit11->Text+"\n");
  }
  for(y=0;y<2;y++)     //kolom
  {
   for(x=0;x<pop+1;x++)  //baris
   {
```

```
    SG_fp->Cells[y][x]=SG_fitness->Cells[y][x];
   }
  }
 }
 else if(t>0)
 {
  for(a=1;a<pop+1;a++)   //baris
  {
    //ShowMessage("baris ke-"+String(a));
    if(StrToInt(SG_fitness->Cells[1][a])<StrToInt(SG_fp1->Cells[1][a]))
    {
      // ShowMessage("update");
       for(y=0;y<2;y++)
       {
          SG_fp->Cells[y][a]=SG_fitness->Cells[y][a];
       }
       for(j=1;j<job+1;j++)
       {
          SG_personal->Cells[j][a]=SG_populasi->Cells[j][a];
       }
    }
  }
   jumjob=2*job;
    for(i=1;i<pop+1;i++)
    {
     Edit19->Text="  ";
     Edit20->Text="  ";
     for(j=1;j<=jumjob;j++)
     {

       Edit19->Text=Edit19->Text+"  "+String(SG_personal->Cells[j][i]);
       if(j>job)
       {
        Edit20->Text=Edit20->Text+"  "+String(SG_plama->Cells[j-job][i]);
       }
     }
     RE_proses->Lines->Add("Particle  "+String(i)+"\n"+String(Edit19->Text)+"  <--
"+String(Edit20->Text)+"\n") ;
    }
 }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::urut2()
{
  SG_urut2->ColCount=2;
  SG_urut2->RowCount=pop+1;

  for(x=0;x<SG_urut2->RowCount;x++)
  {
    for(y=0;y<2;y++)
    {
         SG_urut2->Cells[y][x]=SG_fp->Cells[y][x];
    }
  }
  for(x=1;x<=pop-1;x++)
  {
   for(x1=x+1;x1<=pop;x1++)
   {
     data1=StrToInt(SG_urut2->Cells[1][x]);
     data2=StrToInt(SG_urut2->Cells[1][x1]);
     if(data1>data2)
     {
        for(k=0;k<SG_urut2->ColCount;k++)//kolom
         {
           data=StrToInt(SG_urut2->Cells[k][x1]);
           SG_urut2->Cells[k][x1]=StrToInt(SG_urut2->Cells[k][x]);
           SG_urut2->Cells[k][x]=data;
         }
     }
   }
  }
```

```
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Global_Best(int t)
{
  SG_global->ColCount=job+1;
  SG_global->RowCount=3;
  SG_fglobal->ColCount=2;
  SG_fglobal->RowCount=3;
  SG_globaljob->ColCount=job+1;
  SG_globaljob->RowCount=3;
  urut2();
  RE_proses->Lines->Add("\n========== Global Best ==========");
  if(t==0)
  {
    SG_fglobal->Cells[0][1]=SG_urut2->Cells[0][1];
    SG_fglobal->Cells[1][1]=SG_urut2->Cells[1][1];
    //mencari letak posisi global best
    particle=StrToInt(SG_urut2->Cells[0][1]);
    Edit12->Text=" ";
    for(l=1;l<job+1;l++)
    {
     SG_global->Cells[l][1]=SG_personal->Cells[l][particle];
     SG_globaljob->Cells[l][1]=SG_job->Cells[l][particle];
     Edit12->Text=Edit12->Text+"  "+String(SG_global->Cells[l][1]);
    }
    RE_proses->Lines->Add("  "+Edit12->Text);

  }
  else if(t>0)
  {
    if(StrToInt(SG_urut2->Cells[1][1])<StrToInt(SG_fglobal->Cells[1][2]))
    {
      //update global best
      SG_fglobal->Cells[0][1]=SG_urut2->Cells[0][1];
      SG_fglobal->Cells[1][1]=SG_urut2->Cells[1][1];
      //mencari letak posisi global best
       particle=StrToInt(SG_urut2->Cells[0][1]);
       for(l=1;l<job+1;l++)
       {
          SG_global->Cells[l][1]=SG_personal->Cells[l][particle];
          SG_globaljob->Cells[l][1]=SG_job->Cells[l][particle];
       }
    }

    else
    {
      SG_fglobal->Cells[1][1]=SG_fglobal->Cells[1][2];
      //mencari letak posisi global best
      for(l=1;l<job+1;l++)
      {
         SG_global->Cells[l][1]=SG_global->Cells[l][2];
         SG_globaljob->Cells[l][1]=SG_globaljob->Cells[l][2];
      }
    }
    Edit21->Text="  ";
    for(l=1;l<job+1;l++)
    {
     Edit21->Text=Edit21->Text+"  "+String(SG_global->Cells[l][1]);
    }
    RE_proses->Lines->Add(" "+Edit21->Text);
  }

}
//---------------------------------------------------------------------------

void __fastcall TFormProses::Inertia_Weight()
{
 SG_w->RowCount=maxiter;
 SG_w->ColCount=2;
 maxiter=StrToInt(FormInisialisasi->Edit_maxiter->Text);
 w0=StrToFloat(FormInisialisasi->Edit_w->Text);
```

```
 alpha=StrToFloat(FormInisialisasi->Edit_alpha->Text);
 for(z=0;z<maxiter;z++)
 {
  SG_w->Cells[0][z]="w"+String(z);
 }

 SG_w->Cells[1][0]=w0;
 for(z=1;z<maxiter;z++)
 {

    inertia=StrToFloat(SG_w->Cells[1][z-1]);
    SG_w->Cells[1][z]=(inertia*alpha);
 }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Update_Velocity(int t)
{
  SG_vellama->RowCount=pop+1;
  SG_vellama->ColCount=job+1;
  SG_velbaru->RowCount=pop+1;
  SG_velbaru->ColCount=job+1;
  SG_plama->RowCount=pop+1;
  SG_plama->ColCount=job+1;
  SG_global->ColCount=job+1;
  SG_global->RowCount=3;
  c1=StrToFloat(FormInisialisasi->Edit_c1->Text);
  c2=StrToFloat(FormInisialisasi->Edit_c2->Text);
  w0=StrToFloat(FormInisialisasi->Edit_w->Text);
  alpha=StrToFloat(FormInisialisasi->Edit_alpha->Text);
  vmax=StrToFloat(FormInisialisasi->Edit_vmax->Text);
  vmin=-vmax;
  acak=random(10000);
  acak1=random(10000);
  r1=acak/10000;
  r2=acak1/10000;
  Inertia_Weight();
  w=StrToFloat(SG_w->Cells[1][t-1]);
  RE_proses->Lines->Add("\n========== Velocity ==========");
  for(ii=1;ii<pop+1;ii++)
  {
   Edit13->Text=" ";
   for(jj=1;jj<job+1;jj++)
   {
    velocitylama=StrToFloat(SG_vellama->Cells[jj][ii]);
    personal=StrToFloat(SG_plama->Cells[jj][ii]);
    global=StrToFloat(SG_global->Cells[jj][2]);
    posisi=StrToFloat(SG_poplama->Cells[jj][ii]);
    SG_velbaru->Cells[jj][ii]=(w*velocitylama)+(c1*r1*(personal-
posisi))+(c2*r2*(global-posisi));
    if(StrToFloat(SG_velbaru->Cells[jj][ii])>vmax)
     {
      SG_velbaru->Cells[jj][ii]=vmax;
     }
     if(StrToFloat(SG_velbaru->Cells[jj][ii])<vmin)
     {
      SG_velbaru->Cells[jj][ii]=vmin;
     }
     Edit13->Text=Edit13->Text+"  "+String(SG_velbaru->Cells[jj][ii]);
   }
   RE_proses->Lines->Add("Velocity  untuk  particle  "+String(ii)+"    \n"+Edit13-
>Text+"\n");
  }

}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Simpan()
{
  //ShowMessage("Simpan");
  SG_vellama->RowCount=pop+1;
  SG_vellama->ColCount=job+1;
  SG_plama->RowCount=pop+1;
```

```
  SG_plama->ColCount=job+1;
  SG_poplama->RowCount=pop+1;
  SG_poplama->ColCount=job+1;
  SG_velocity->RowCount=pop+1;
  SG_velocity->ColCount=job+1;
  SG_populasi->RowCount=pop+1;
  SG_populasi->ColCount=job+1;
  SG_personal->RowCount=pop+1;
  SG_personal->ColCount=job+1;
  SG_joblama->RowCount=pop+1;
  SG_joblama->ColCount=job+1;
  SG_job->RowCount=pop+1;
  SG_job->ColCount=job+1;
  SG_urut2->RowCount=pop+1;
  SG_urut2->ColCount=2;
  SG_urut2lama->RowCount=pop+1;
  SG_urut2lama->ColCount=2;
  SG_fp1->RowCount=pop+1;
  SG_fp1->ColCount=2;
  SG_fp->RowCount=pop+1;
  SG_fp->ColCount=2;
  SG_global->RowCount=3;
  SG_global->ColCount=job+1;
  SG_globaljob->RowCount=3;
  SG_globaljob->ColCount=job+1;
  SG_fglobal->RowCount=3;
  SG_fglobal->ColCount=2;

  for(g=0;g<pop+1;g++)
  {
   for(h=0;h<job+1;h++)
   {
     SG_poplama->Cells[h][g]=SG_populasi->Cells[h][g];
     SG_vellama->Cells[h][g]=SG_velocity->Cells[h][g];
     SG_joblama->Cells[h][g]=SG_job->Cells[h][g];
     SG_plama->Cells[h][g]=SG_personal->Cells[h][g];
   }
  }
  for(g=0;g<pop+1;g++)
  {
    for(a=0;a<2;a++)
    {
      SG_urut2lama->Cells[a][g]=SG_urut2->Cells[a][g];
      SG_fp1->Cells[a][g]=SG_fp->Cells[a][g];
    }
  }
  SG_fglobal->Cells[0][2]=SG_fglobal->Cells[0][1];
  SG_fglobal->Cells[1][2]=SG_fglobal->Cells[1][1];
  for(h=0;h<job+1;h++)
  {
    SG_global->Cells[h][2]=SG_global->Cells[h][1];
    SG_globaljob->Cells[h][2]=SG_globaljob->Cells[h][1];
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Update_Populasi()
{
  SG_popbaru->RowCount=pop+1;
  SG_popbaru->ColCount=job+1;
  RE_proses->Lines->Add("\n========== Populasi ==========");
  for(i=1;i<pop+1;i++)
  {
   Edit14->Text=" ";
   for(j=1;j<job+1;j++)
   {
     velocitybaru=StrToFloat(SG_velbaru->Cells[j][i]);
     posisi1=StrToFloat(SG_poplama->Cells[j][i]);
     SG_popbaru->Cells[j][i]=velocitybaru+posisi1;
     Edit14->Text=Edit14->Text+"  "+String(SG_popbaru->Cells[j][i]);
   }
   RE_proses->Lines->Add("Particle "+String(i)+"\n"+Edit14->Text+"\n");
```

```
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Simpan1()
{
  for(i=1;i<pop+1;i++)
  {
   for(j=1;j<job+1;j++)
   {
     SG_populasi->Cells[j][i]=SG_popbaru->Cells[j][i];
     SG_velocity->Cells[j][i]=SG_velbaru->Cells[j][i];
   }
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Ambil_random()
{
  Edit23->Text="  ";
  Edit24->Text="  ";
  kappa=random(job);
  kappa1=kappa+1;
  do
  {
    tetha=random(job);
    tetha1=tetha+1;
   }
   while(tetha1==kappa1);
  Edit23->Text=Edit23->Text+" "+String(kappa1);
  Edit24->Text=Edit24->Text+" "+String(tetha1);
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Insert()
{
  RE_proses->Lines->Add("-----Fungsi Insert-----");
  //ShowMessage("Insert");
  Ambil_random();
  RE_proses->Lines->Add("Kappa      :      "+String(Edit23->Text)+"\nTetha      :
"+String(Edit24->Text));
  SG_local->ColCount=job+1;
  SG_local->RowCount=3;
  if(tetha1<kappa1)
  {
    selisih=kappa1-tetha1;
    if(selisih!=1)
    {
      simpan=StrToInt(SG_local->Cells[tetha1][0]);
      for(ii=tetha1;ii<=kappa1-2;ii++) //kolom
      {
        SG_local->Cells[ii][0]=SG_local->Cells[ii+1][0];
      }
      SG_local->Cells[kappa1-1][0]=simpan;
    }
  }
  else
  {
   selisih1=tetha1-kappa1;
   if(selisih1!=1)
   {
     simpan=StrToInt(SG_local->Cells[tetha1][0]);
     {
      for(ii=tetha1;ii>=kappa1+2;ii--) //kolom
      {
        SG_local->Cells[ii][0]=SG_local->Cells[ii-1][0];
      }
      SG_local->Cells[kappa1+1][0]=simpan;
     }
   }

  }
}
//---------------------------------------------------------------------------
```

```
void __fastcall TFormProses::Interchange()
{
  //ShowMessage("Interchange");
  RE_proses->Lines->Add("-----Fungsi Interchange------");
  Ambil_random();
  RE_proses->Lines->Add("Kappa      :      "+String(Edit23->Text)+"\nTetha      :
"+String(Edit24->Text));
  SG_local->ColCount=job+1;
  SG_local->RowCount=3;

  simpan1=StrToInt(SG_local->Cells[tetha1][0]);
  SG_local->Cells[tetha1][0]=SG_local->Cells[kappa1][0];
  SG_local->Cells[kappa1][0]=simpan1;
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Evaluasi1()
{
  SG_nilailocal->ColCount=2;
  SG_nilailocal->RowCount=3;

  for(i=1;i<=job;i++)
  {
   for(j=1;j<=job;j++)
   {
     if(StrToInt(SG_local->Cells[i][0])==j)
     {
       for(k=1;k<=mesin;k++)
       {
         SG_jobtime->Cells[0][i]=SG_copy1->Cells[0][j];
         SG_jobtime->Cells[k][i]=SG_copy1->Cells[k][j];
       }
     }
   }
  }
  Makespan();

  SG_nilailocal->Cells[1][0]=nilai;

}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Local_search()
{
   RE_proses->Lines->Add("\n==========  Local  Search  ==========\n          @@@@@
Prosedur Local Search VNS @@@@@");
   SG_local->Cells[0][1]="S";
   SG_local->Cells[0][2]="S1";
   SG_nilailocal->Cells[0][1]="f(S)";
   SG_nilailocal->Cells[0][2]="f(S1)";
   SG_local->Cells[0][0]="y";
   SG_nilailocal->Cells[0][0]="f(y)";
   //y=global
   Edit22->Text=" ";
   for(jj=1;jj<=job;jj++)
   {
     SG_local->Cells[jj][0]=SG_globaljob->Cells[jj][1];
     Edit22->Text=Edit22->Text+"  "+String(SG_local->Cells[jj][0]);
     //SG_local->Cells[jj][0]=SG_local->Cells[jj][1];
   }
   RE_proses->Lines->Add("Job Sequence Awal : "+String(Edit22->Text));
   Insert();
   Evaluasi1();
   Edit27->Text=" ";
   SG_nilailocal->Cells[1][1]=SG_nilailocal->Cells[1][0];
   Edit27->Text=Edit27->Text+" "+String(SG_nilailocal->Cells[1][1]);
   //S=y
   Edit25->Text=" ";
   for(jj=1;jj<=job;jj++)
   {
     SG_local->Cells[jj][1]=SG_local->Cells[jj][0];
     Edit25->Text=Edit25->Text+"  "+String(SG_local->Cells[jj][1]);
   }
```

```
   RE_proses->Lines->Add("Job  Sequence  :  "+String(Edit25->Text)+"\nMakespan  :
"+String(Edit27->Text));
   for(loop=0;loop<(job*(job-1));loop++)
   {
     RE_proses->Lines->Add("\n***Looping ke- "+String(loop)+"***");
     //ShowMessage("Loop "+String(loop));

     for(kcount=0;kcount<2;kcount++)
     {
       //ShowMessage("Kcount "+String(kcount));
      //S1=S
      //y=S1
       for(jj=1;jj<=job;jj++)
       {
         SG_local->Cells[jj][2]=SG_local->Cells[jj][1];
         SG_local->Cells[jj][0]=SG_local->Cells[jj][2];
       }
       if(kcount==0)
       {
         Insert();
       }
       else if(kcount==1)
       {
         Interchange();
       }
         Edit26->Text=" ";
         Evaluasi1();
         //f(s1)=f(y)
         SG_nilailocal->Cells[1][2]=SG_nilailocal->Cells[1][0];
         Edit26->Text=Edit26->Text+String(SG_nilailocal->Cells[1][2]);
         Edit28->Text=" ";
         for(jj=1;jj<=job;jj++)
         {
          SG_local->Cells[jj][2]=SG_local->Cells[jj][0];
          Edit28->Text=Edit28->Text+"  "+String(SG_local->Cells[jj][2]);
         }
         RE_proses->Lines->Add("Job  Sequence  :  "+String(Edit28->Text)+"\nMakespan
: "+String(Edit26->Text));
       //f(s)>f(s1)
       if(StrToInt(SG_nilailocal->Cells[1][1])>StrToInt(SG_nilailocal-
>Cells[1][2]))
       {
          kcount=0;
          //s=s1
          for(jj=1;jj<=job;jj++)
          {
            SG_local->Cells[jj][1]=SG_local->Cells[jj][2];
          }
          //f(s)=f(s1)
          SG_nilailocal->Cells[1][1]=SG_nilailocal->Cells[1][2];
       }

     }
   }
   RE_proses->Lines->Add("\n*******Hasil dari Local Search VNS*******");
   if(StrToInt(SG_nilailocal->Cells[1][1])<=StrToInt(SG_fglobal->Cells[1][1]))
   {
     //global=s
     for(jj=1;jj<=job;jj++)
     {
       SG_globaljob->Cells[jj][1]=SG_local->Cells[jj][1];
     }
     //f(global)=f(s)
     SG_fglobal->Cells[1][1]=SG_nilailocal->Cells[1][1];
    //perbaiki global best
     Perbaiki();
   }
   Edit29->Text=" ";
   Edit30->Text=" ";
   Edit31->Text="  ";
   for(jj=1;jj<=job;jj++)
```

```
   {
    Edit29->Text=Edit29->Text+"  "+String(SG_globaljob->Cells[jj][1]);
    Edit30->Text=Edit30->Text+"  "+String(SG_global->Cells[jj][1]);
   }
   Edit31->Text=Edit31->Text+String(SG_fglobal->Cells[1][1]);
   RE_proses->Lines->Add("Global Best : "+String(Edit30->Text)+"\nJob Sequence :
"+String(Edit29->Text)+"\nMakespan : "+String(Edit31->Text));

}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Perbaiki()
{
  SG_simpan->ColCount=job;
  SG_simpan->RowCount=1;

  for(i=0;i<job;i++)
  {
   SG_simpan->Cells[i][0]=SG_global->Cells[i+1][1];
  }
  // bubble short
  for(j=0;j<job-1;j++)
  {
    for(k=j+1;k<job;k++)
    {
      if(StrToFloat(SG_simpan->Cells[j][0])>StrToFloat(SG_simpan->Cells[k][0]))
      {
        tmp1=StrToFloat(SG_simpan->Cells[j][0]);
        SG_simpan->Cells[j][0]=SG_simpan->Cells[k][0];
        SG_simpan->Cells[k][0]=tmp1;
      }
    }
  }
  for(kk=1;kk<=job;kk++)
  {
   for(ii=1;ii<=job;ii++)
   {
     urutan=StrToInt(SG_globaljob->Cells[kk][1]);
     if(urutan==ii)
     {
       SG_global->Cells[urutan][1]=SG_simpan->Cells[kk-1][0];
     }
   }
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Hasil(int t)
{
  SG_iterasimakespan->RowCount=maxiter+1;
  SG_iterasimakespan->ColCount=2;
  SG_iterasiposisi->RowCount=maxiter+1;
  SG_iterasiposisi->ColCount=job+1;
  SG_iterasijob->RowCount=maxiter+1;
  SG_iterasijob->ColCount=job+1;

  SG_iterasimakespan->Cells[0][t]="Iterasi "+String(t);
  SG_iterasiposisi->Cells[0][t]="Iterasi "+String(t);
  SG_iterasijob->Cells[0][t]="Iterasi "+String(t);

  SG_iterasimakespan->Cells[1][t]=SG_fglobal->Cells[1][1];
  for(c=1;c<job+1;c++)
  {
   SG_iterasiposisi->Cells[c][t]=SG_global->Cells[c][1];
   SG_iterasijob->Cells[c][t]=SG_globaljob->Cells[c][1];
  }
  Edit1->Text=" ";
  Edit2->Text=" ";
  if(t==maxiter)
  {
    for(j=1;j<job+1;j++)
    {
     Edit1->Text=Edit1->Text +" "+String(SG_iterasijob->Cells[j][maxiter]);
```

```
    }
    Edit2->Text=Edit2->Text +" "+String(SG_iterasimakespan->Cells[1][maxiter]);
  }
}
//---------------------------------------------------------------------------

void __fastcall TFormProses::PSO()
{
  maxiter=StrToInt(FormInisialisasi->Edit_maxiter->Text);
  for(t=0;t<=maxiter;t++)
  {
   if(t==0)
   {
    RE_proses->Lines->Add("\nxxxxxxxxx Iterasi Inisialisasi xxxxxxxxx");
    Populasi_awal();
    for(i=1;i<=pop;i++)
    {
     Edit5->Text=" ";
     for(j=1;j<=job;j++)
     {
      Edit5->Text=Edit5->Text+"  "+String(SG_populasi->Cells[j][i]);
     }
     RE_proses->Lines->Add("Particle "+String(i)+" \n"+Edit5->Text+"\n");
    }
    Velocity_awal();
    for(i=1;i<=pop;i++)
    {
     Edit6->Text=" ";
     for(j=1;j<=job;j++)
     {
      Edit6->Text=Edit6->Text+"  "+String(SG_velocity->Cells[j][i]);
     }
     RE_proses->Lines->Add("Velocity   untuk   particle   "+String(i)+"   \n"+Edit6-
>Text+"\n");
    }
    Permutasi();
    for(i=1;i<=pop;i++)
    {
      Edit7->Text=" ";
      Edit8->Text=" ";
      for(j=1;j<=job;j++)
      {
        Edit7->Text=Edit7->Text+"  "+String(SG_populasi->Cells[j][i]);
        Edit8->Text=Edit8->Text+"  "+String(SG_job->Cells[j][i]);
      }
      RE_proses->Lines->Add("Posisi  "+String(i)+"\n"+Edit7->Text+"\nJob  Sequence
:"+String(Edit8->Text)+"\n");
    }
    Evaluasi();
    for(i=1;i<=pop;i++)
    {
     Edit10->Text=" ";
     Edit9->Text=" ";

     for(j=1;j<=job;j++)
     {
      Edit9->Text=Edit9->Text+"  "+String(SG_job->Cells[j][i]);
     }
     Edit10->Text=Edit10->Text+"  "+String(SG_fitness->Cells[1][i]);
     RE_proses->Lines->Add("Particle "+String(i)+"\nJob Sequence : "+String(Edit9-
>Text)+"\nMakespan : "+String(Edit10->Text)+"\n");
    }
    Personal_Best(t);
    Global_Best(t);
    Hasil(t);
    Simpan();
   }
   else if(t>0)
   {
    RE_proses->Lines->Add("\nxxxxxxxxxxxxx          Iterasi          "+String(t)+"
xxxxxxxxxxxxxx");
```

```
    Update_Velocity(t);
    Update_Populasi();
    Simpan1();
    Permutasi();
    for(i=1;i<=pop;i++)
    {
      Edit15->Text=" ";
      Edit16->Text=" ";
      for(j=1;j<=job;j++)
      {
        Edit15->Text=Edit15->Text+"  "+String(SG_populasi->Cells[j][i]);
        Edit16->Text=Edit16->Text+"  "+String(SG_job->Cells[j][i]);
      }
      RE_proses->Lines->Add("Posisi  "+String(i)+"\n"+Edit15->Text+"\nJob  Sequence
:"+String(Edit16->Text)+"\n");
    }
    Evaluasi();
     for(i=1;i<=pop;i++)
    {
     Edit17->Text=" ";
     Edit18->Text=" ";

     for(j=1;j<=job;j++)
     {
      Edit17->Text=Edit17->Text+"  "+String(SG_job->Cells[j][i]);
     }
     Edit18->Text=Edit18->Text+"  "+String(SG_fitness->Cells[1][i]);
     RE_proses->Lines->Add("Particle      "+String(i)+"\nJob      Sequence      :
"+String(Edit17->Text)+"\nMakespan : "+String(Edit18->Text)+"\n");
    }
    Personal_Best(t);
    Global_Best(t);
    Local_search();
    Hasil(t);
    Simpan();
   }
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Mean()
{
  SG_hasil->ColCount=2;
  SG_hasil->RowCount=2;
  iterasi=StrToFloat(FormInisialisasi->Edit_maxiter->Text);
  jummakespan=0;
  for(i=0;i<=maxiter;i++)
  {
    makespan=StrToInt(SG_iterasimakespan->Cells[1][i]);
    jummakespan=jummakespan+makespan;
  }
  mean=jummakespan/(iterasi+1);
  Edit3->Text=" ";
  SG_hasil->Cells[0][0]="Mean";
  SG_hasil->Cells[1][0]=mean;
  Edit3->Text=Edit3->Text+" "+String(SG_hasil->Cells[1][0]);
}
//---------------------------------------------------------------------------
void __fastcall TFormProses::Standart_deviasi()
{
  SG_hasil->ColCount=2;
  SG_hasil->RowCount=2;
  iterasi=StrToFloat(FormInisialisasi->Edit_maxiter->Text);
  sigma=0;
  for(j=0;j<=maxiter;j++)
  {
   xi=StrToFloat(SG_iterasimakespan->Cells[1][j]);
   sigma=sigma+((xi-mean)*(xi-mean));
  }
  SD=sqrt(sigma/(iterasi));
  Edit4->Text=" ";
  SG_hasil->Cells[0][1]="Standart Deviasi";
```

```
  SG_hasil->Cells[1][1]=SD;
  Edit4->Text=Edit4->Text+" "+String(SG_hasil->Cells[1][1]);
}

//---------------------------------------------------------------------------
void __fastcall TFormProses::BitBtn_prosesClick(TObject *Sender)
{
  RE_hasil->Lines->Clear();
  RE_hasil->Lines->Add("***** Aplikasi C++ Builder *****");
  RE_hasil->Lines->Add("ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN
   PENJADWALAN PERMUTATION FLOWSHOP");
  RE_hasil->Lines->Add("\n=== Permasalahan yang akan diselesaikan ===");
  RE_hasil->Lines->Add("Jumlah Job     : "+ String(job));
  RE_hasil->Lines->Add("Jumlah Mesin   : "+ String(mesin));
  RE_hasil->Lines->Add("Jumlah Iterasi : "+ String(FormInisialisasi->Edit_maxiter-
>Text));
  RE_hasil->Lines->Add("Jumlah  Particle  :  "+  String(FormInisialisasi->Edit_pop-
>Text));
  RE_hasil->Lines->Add("Xmax : "+ String(FormInisialisasi->Edit_xmax->Text));
  RE_hasil->Lines->Add("Vmax : "+ String(FormInisialisasi->Edit_vmax->Text));
  RE_hasil->Lines->Add("\n=== Parameter yang digunakan ===");
  RE_hasil->Lines->Add("c1      : "+ String(FormInisialisasi->Edit_c1->Text));
  RE_hasil->Lines->Add("c2      : "+ String(FormInisialisasi->Edit_c2->Text));
  RE_hasil->Lines->Add("w0     : "+ String(FormInisialisasi->Edit_w->Text));
  RE_hasil->Lines->Add("Alpha  : "+ String(FormInisialisasi->Edit_alpha->Text));
  RE_hasil->Lines->Add("\n=== Solusi yang diperoleh ===");
  RE_proses->Lines->Clear();
  RE_proses->Lines->Add("=== Prosedur PSO dengan Local Search ===");
  randomize();
  PSO();
  RE_hasil->Lines->Add("Job Sequence  : "+ Edit1->Text);
  RE_hasil->Lines->Add("Makespan      : "+ Edit2->Text);
  Mean();
  Standart_deviasi();
  RE_hasil->Lines->Add("Mean Makespan             : "+ Edit3->Text);
  RE_hasil->Lines->Add("Standart Deviasi Makespan  : "+ Edit4->Text);
}
//---------------------------------------------------------------------------

void __fastcall TFormProses::BitBtn_printClick(TObject *Sender)
{
   try
   {
     if(FormPrint1 != NULL)
     {
       FormProses->WindowState=wsNormal;
       FormPrint1->Show();
     }
     else
     {
       FormPrint1=new TFormPrint1(this);
       FormPrint1->Show();
       FormPrint1->WindowState=wsNormal;
     }
   }
   catch(Exception &exception)
   {
     FormPrint1= new TFormPrint1(this);
     FormPrint1->Show();
     FormPrint1->WindowState=wsNormal;
   }
}
//---------------------------------------------------------------------------

void __fastcall TFormProses::BitBtn_saveClick(TObject *Sender)
{
  try
  {
    if(FormSave1 != NULL)
    {
      FormProses->WindowState=wsNormal;
```

```
      FormSave1->Show();
    }
    else
    {
      FormSave1=new TFormSave1(this);
      FormSave1->Show();
      FormSave1->WindowState=wsNormal;
    }
  }
  catch(Exception &exception)
  {
    FormSave1=new TFormSave1(this);
    FormSave1->Show();
    FormSave1->WindowState=wsNormal;
  }
}
//---------------------------------------------------------------------------

void __fastcall TFormProses::BitBtn_ganttchartClick(TObject *Sender)
{
  try
  {
    if(FormGantt_Chart != NULL)
    {
     FormProses->WindowState=wsMinimized;
     FormGantt_Chart->Show();
    }
    else
    {
     FormProses->WindowState=wsMinimized;
     FormGantt_Chart=new TFormGantt_Chart(this);
     FormGantt_Chart->Show();
     FormGantt_Chart->WindowState=wsMaximized;
    }
  }
  catch(Exception &exception)
  {
   FormProses->WindowState=wsMinimized;
   FormGantt_Chart=new TFormGantt_Chart(this);
   FormGantt_Chart->Show();
   FormGantt_Chart->WindowState=wsMaximized;
  }
}
//---------------------------------------------------------------------------
```

gantt_chart.h

```
//---------------------------------------------------------------------------

#ifndef gantt_chartH
#define gantt_chartH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <Grids.hpp>
//---------------------------------------------------------------------------
class TFormGantt_Chart : public TForm
{
__published:   // IDE-managed Components
        TGroupBox *GroupBox1;
        TStringGrid *SG_ganttchart;
        TBitBtn *BitBtn_ganttchart;
        TStringGrid *SG_jobtime1;
        TStringGrid *SG_start1;
```

```
        TStringGrid *SG_end1;
        void __fastcall BitBtn_ganttchartClick(TObject *Sender);
        void __fastcall hitung();
private:       // User declarations
public:        // User declarations
        __fastcall TFormGantt_Chart(TComponent* Owner);
        int job,mesin,i,j,k,h,a,b,c,d,e,maxiter;
};
//---------------------------------------------------------------------------
extern PACKAGE TFormGantt_Chart *FormGantt_Chart;
//---------------------------------------------------------------------------
#endif
```

gantt_chart.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

#include "opening.h"
#include "gantt_chart.h"
#include "proses.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormGantt_Chart *FormGantt_Chart;
//---------------------------------------------------------------------------
__fastcall TFormGantt_Chart::TFormGantt_Chart(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormGantt_Chart::BitBtn_ganttchartClick(TObject *Sender)
{
  job=StrToInt(FormProses->job);
  mesin=StrToInt(FormProses->mesin);
  maxiter=StrToInt(FormProses->maxiter);
  a=StrToInt(FormProses->SG_iterasimakespan->Cells[1][maxiter]);
  SG_ganttchart->ColCount=a+1;
  SG_ganttchart->RowCount=mesin+1;

  hitung();
  for(i=0;i<=mesin;i++)
  {
   if(i<mesin)
   {
    SG_ganttchart->Cells[0][i]="M"+String(i+1)+"";
   }
   else
   {
    SG_ganttchart->Cells[0][i]="T";
    for(j=1;j<=a+2;j++)
    {
     SG_ganttchart->Cells[j][i]=j;
    }
   }
  }

   for(i=1;i<=mesin;i++)      //kolom
   {
     for(k=1;k<=job;k++)       //baris
     {
       b=StrToInt(SG_end1->Cells[i][k]);
       d=StrToInt(SG_end1->Cells[0][k]);
       if(b!=0)
       {
```

```
          c=StrToInt(SG_start1->Cells[i][k]);
          for(h=1;h<SG_ganttchart->ColCount;h++)
          {
            if(h>=c+1&&h<=b)
            {
              SG_ganttchart->Cells[h][i-1]=d;
              c++;
            }
          }
        }
       }
     }
}
//---------------------------------------------------------------------------
void __fastcall TFormGantt_Chart::hitung()
{
  job=StrToInt(FormProses->job);
  mesin=StrToInt(FormProses->mesin);
  maxiter=StrToInt(FormProses->maxiter);
  SG_jobtime1->ColCount=mesin+1;
  SG_jobtime1->RowCount=job+1;
  SG_start1->ColCount=mesin+1;
  SG_start1->RowCount=job+1;
  SG_end1->ColCount=mesin+1;
  SG_end1->RowCount=job+1;

  for(i=0;i<job+1;i++)        //baris
  {
   for(j=0;j<mesin+1;j++)     //kolom
   {
   //ShowMessage("i="+String(i)+",j="+String(j));
    if(i==0 && j==0)
    {
     SG_jobtime1->Cells[j][i]="Jobtime";
    }
    else if(i==0 && j!=0)
    {
     SG_jobtime1->Cells[j][i]=j;
    }
    else if(i!=0 && j==0)
    {
     SG_jobtime1->Cells[j][i]=StrToInt(FormProses->SG_iterasijob-
>Cells[i][maxiter]) ;
    }
    else
    {
     for(b=1;b<=job;b++)
     {
      if(StrToInt(SG_jobtime1->Cells[0][i])==StrToInt(FormProses->SG_copy1-
>Cells[0][b]))
      {
        SG_jobtime1->Cells[j][i]=StrToInt(FormProses->SG_copy1->Cells[j][b]);
      }
     }
    }
   // ShowMessage("jobtime="+String(SG_jobtime1->Cells[j][i]));
   }
  }

  for(i=1;i<SG_start1->RowCount;i++)
   {
    for(j=1;j<SG_start1->ColCount;j++)
    {
     SG_start1->Cells[0][i]=SG_jobtime1->Cells[0][i];
     SG_start1->Cells[j][0]=SG_jobtime1->Cells[j][0];
     SG_end1->Cells[0][i]=SG_jobtime1->Cells[0][i];
     SG_end1->Cells[j][0]=SG_jobtime1->Cells[j][0];
    }
   }
   for(i=1;i<=job;i++) //baris
   {
```

```
     for(j=1;j<=mesin;j++) //kolom
     {
       if(i==1 && j==1)
       {
         SG_start1->Cells[j][i]=0;
         SG_end1->Cells[j][i]=SG_jobtime1->Cells[j][i];
       }
       else if(i==1 && j>1)
       {
         SG_start1->Cells[j][i]=SG_end1->Cells[j-1][i];
       }
       else if(i>1 && j==1)
       {
         SG_start1->Cells[j][i]=SG_end1->Cells[j][i-1];
       }
       else if(i>1 && j>1)
       {
         if(StrToInt(SG_end1->Cells[j-1][i])>StrToInt(SG_end1->Cells[j][i-1]))
         {
           SG_start1->Cells[j][i]=SG_end1->Cells[j-1][i];
         }
         else
         {
           SG_start1->Cells[j][i]=SG_end1->Cells[j][i-1];
         }
       }
       SG_end1->Cells[j][i]=StrToInt(SG_start1->Cells[j][i])+StrToInt(SG_jobtime1-
>Cells[j][i]);
     }
   }
}
//---------------------------------------------------------------------------
```

save1.h

```
//---------------------------------------------------------------------------

#ifndef save1H
#define save1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TFormSave1 : public TForm
{
__published:   // IDE-managed Components
        TShape *Shape1;
        TLabel *Label1;
        TBitBtn *BitBtn_ok;
        TBitBtn *BitBtn_cancel;
        TSaveDialog *SD_save1;
        void __fastcall BitBtn_okClick(TObject *Sender);
        void __fastcall BitBtn_cancelClick(TObject *Sender);

private:       // User declarations
public:        // User declarations
        __fastcall TFormSave1(TComponent* Owner);
        String Filename;
};
//---------------------------------------------------------------------------
extern PACKAGE TFormSave1 *FormSave1;
//---------------------------------------------------------------------------
#endif
```

save1.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

#include "proses.h"
#include "save1.h"
#include "save2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormSave1 *FormSave1;
//---------------------------------------------------------------------------
__fastcall TFormSave1::TFormSave1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------


void __fastcall TFormSave1::BitBtn_okClick(TObject *Sender)
{
  if(SD_save1->Execute())
  {
   std::auto_ptr<TStrings>LoadStrings(new TStringList());
   LoadStrings->SaveToFile(SD_save1->FileName);
   Filename=SD_save1->FileName;
   FormProses->RE_hasil->Lines->SaveToFile(Filename);
  }
  FormSave1->WindowState=wsMinimized;
  try
  {
    if(FormSave2 != NULL)
    {
      FormProses->WindowState = wsMaximized;
      FormSave2->Show();
    }
    else
    {
      FormSave2= new TFormSave2(this);
      FormSave2->Show();
      FormSave2->WindowState= wsMinimized;
    }
  }
  catch(Exception &exception)
  {
    FormSave2= new TFormSave2(this);
    FormSave2->Show();
    FormSave2->WindowState=wsMinimized;
  }
}
//---------------------------------------------------------------------------
void __fastcall TFormSave1::BitBtn_cancelClick(TObject *Sender)
{
   FormSave1->WindowState=wsMinimized;
   try
   {
     if(FormSave2 != NULL)
     {
       FormProses->WindowState=wsMaximized;
       FormSave2->Show();
     }
     else
     {
       FormSave2=new TFormSave2(this);
```

```
      FormSave2->Show();
      FormSave2->WindowState=wsNormal;
    }
  }
  catch(Exception &exception)
  {
    FormSave2=new TFormSave2(this);
    FormSave2->Show();
    FormSave2->WindowState=wsNormal;
  }
}
//---------------------------------------------------------------------------
```

save2.h

```
//---------------------------------------------------------------------------

#ifndef save2H
#define save2H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TFormSave2 : public TForm
{
__published:   // IDE-managed Components
        TShape *Shape1;
        TSaveDialog *SD_save2;
        TBitBtn *BitBtn_ok;
        TBitBtn *BitBtn_cancel;
        TLabel *Label1;
        void __fastcall BitBtn_cancelClick(TObject *Sender);
        void __fastcall BitBtn_okClick(TObject *Sender);
private:       // User declarations
public:        // User declarations
        __fastcall TFormSave2(TComponent* Owner);
        String Filename;
};
//---------------------------------------------------------------------------
extern PACKAGE TFormSave2 *FormSave2;
//---------------------------------------------------------------------------
#endif
```

save2.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

#include "proses.h"
#include "save1.h"
#include "save2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormSave2 *FormSave2;
//---------------------------------------------------------------------------
__fastcall TFormSave2::TFormSave2(TComponent* Owner)
        : TForm(Owner)
{
```

```
}
//---------------------------------------------------------------------------
void __fastcall TFormSave2::BitBtn_cancelClick(TObject *Sender)
{
   FormSave1->WindowState = wsNormal;
   if(FormProses !=NULL)
   {
     FormProses->Show();
     FormProses->WindowState = wsMaximized;
   }
}
//---------------------------------------------------------------------------
void __fastcall TFormSave2::BitBtn_okClick(TObject *Sender)
{
  if(SD_save2->Execute())
    {
      std::auto_ptr<TStrings> LoadStrings(new TStringList());
      LoadStrings->SaveToFile(SD_save2->FileName);
      Filename=SD_save2->FileName;
      FormProses->RE_proses->Lines->SaveToFile(Filename);
    }
    FormSave2->WindowState = wsNormal;
      if(FormProses != NULL)
      {
        FormProses->Show();
        FormProses->WindowState = wsMaximized;
      }
}
//---------------------------------------------------------------------------
```

print1.h

```
//---------------------------------------------------------------------------

#ifndef print1H
#define print1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TFormPrint1 : public TForm
{
__published:   // IDE-managed Components
        TLabel *Label1;
        TBitBtn *BitBtn_ok;
        TBitBtn *BitBtn_cancel;
        TShape *Shape1;
        TPrintDialog *PD_print1;
        void __fastcall BitBtn_cancelClick(TObject *Sender);
        void __fastcall BitBtn_okClick(TObject *Sender);
private:       // User declarations
public:        // User declarations
        __fastcall TFormPrint1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFormPrint1 *FormPrint1;
//---------------------------------------------------------------------------
#endif
```

print1.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
```

```
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

#include "proses.h"
#include "print1.h"
#include "print2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormPrint1 *FormPrint1;
//---------------------------------------------------------------------------
__fastcall TFormPrint1::TFormPrint1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormPrint1::BitBtn_cancelClick(TObject *Sender)
{
  try
  {
   if(FormPrint2 !=NULL)
   {
    FormProses->WindowState= wsMaximized;
    FormPrint2->Show();
   }
   else
   {
    FormProses->WindowState = wsMinimized;
    FormPrint2 = new TFormPrint2(this);
    FormPrint2->Show();
    FormPrint2->WindowState = wsNormal;
   }
  }
   catch (Exception &exception)
   {
     FormPrint2 = new TFormPrint2(this);
     FormPrint2->Show();
     FormPrint2->WindowState = wsNormal;
   }
}
//---------------------------------------------------------------------------
void __fastcall TFormPrint1::BitBtn_okClick(TObject *Sender)
{
  if(PD_print1->Execute())
  {
    FormProses->RE_hasil->SelectAll();
    FormProses->RE_hasil->Print(PD_print1->Name);
  }
  FormPrint1->WindowState=wsMinimized;
  try
  {
   if(FormPrint2 != NULL)
   {
    FormProses->WindowState = wsNormal;
    FormPrint2->Show();
   }
   else
   {
    FormPrint2=new TFormPrint2(this);
    FormPrint2->Show();
    FormPrint2->WindowState=wsNormal;
   }
  }
  catch (Exception &exception)
  {
    FormPrint2=new TFormPrint2(this);
    FormPrint2->Show();
    FormPrint2->WindowState=wsNormal;
  }
```

```
}
//---------------------------------------------------------------------------
```

print2.h

```
//---------------------------------------------------------------------------

#ifndef print2H
#define print2H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TFormPrint2 : public TForm
{
__published:   // IDE-managed Components
        TShape *Shape1;
        TLabel *Label1;
        TBitBtn *BitBtn_ok;
        TBitBtn *BitBtn_cancel;
        TPrintDialog *PD_print2;
        void __fastcall BitBtn_cancelClick(TObject *Sender);
        void __fastcall BitBtn_okClick(TObject *Sender);
private:       // User declarations
public:        // User declarations
        __fastcall TFormPrint2(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TFormPrint2 *FormPrint2;
//---------------------------------------------------------------------------
#endif
```

print2.cpp

```
//---------------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include <iostream.h>
#include <conio.h>
#include <stdio.h>

#include "proses.h"
#include "print1.h"
#include "print2.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFormPrint2 *FormPrint2;
//---------------------------------------------------------------------------
__fastcall TFormPrint2::TFormPrint2(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TFormPrint2::BitBtn_cancelClick(TObject *Sender)
{
   FormPrint2->WindowState=wsMinimized;
   if(FormProses != NULL)
   {
     FormProses->Show();
     FormProses->WindowState= wsNormal;
   }
}
```

```
//---------------------------------------------------------------------------
void __fastcall TFormPrint2::BitBtn_okClick(TObject *Sender)
{
  if(PD_print2->Execute())
  {
   FormProses->RE_proses->SelectAll();
   FormProses->RE_proses->Print(PD_print2->Name);
  }
  FormPrint2->WindowState=wsMinimized;
  if(FormProses != NULL)
  {
   FormProses->Show();
   FormProses->WindowState=wsNormal;
  }
}
//---------------------------------------------------------------------------
```

Lampiran 3 : Rincian Hasil Implementasi Program Untuk Permasalahan 4-*Job* 3-Mesin

=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 4
Jumlah Mesin : 3
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1 : 2
C2 : 2
w0 : 0,8999999976158142
Alpha : 0,9499999880079071
=== Solusi yang diperoleh ===
Job Sequence : 2 3 1 4
Makespan : 62

=== Prosedur PSO dengan Local Search ===
xxxxxxxxx Iterasi Inisialisasi xxxxxxxxx
========== Populasi Awal ==========
Particle 1
2,0064001083374 3,730000001907349 1,42879998683929 2,55040001869202
Particle 2
2,71399998664856 0,8191999793052670 0,5483999848365780 1,61199998855591
Particle 3
3,26239991188049 1,72200000286102 3,361200009422302 2,28119993209839
Particle 4
1,90439999103546 3,79959988594055 2,152400016784670 3,67440009117126
Particle 5
0,3991999924182890 2,26799988746643 0,9251999855041500 0,1067999973893170
Particle 6
3,5532000006484990 2,69079995155334 1,8007999665858460 0,0860000029206276
Particle 7
3,6916000843048100 0,6620000004768370 2,8699998855590800 0,6287999749183650
Particle 8
3,1412000656127900 2,9231998920440700 1,7691999673843400 0,3583999872207640
Particle 9
1,6967999935150100 2,6640000343322800 2,6947999000549300 1,3739999532699600
Particle 10
3,8540000915527300 0,0640000030398369 1,7588000297546400 2,0659999847412100
Particle 11
1,4207999706268300 1,1699999570846600 3,7695999145507800 2,5564000606536900
Particle 12
2,3487999439239500 0,5475999712944030 2,1740000247955300 1,1055999994278000
Particle 13
2,8027999401092500 0,6808000206947330 3,4444000720977800 3,2883999347686800
Particle 14
2,7695999145507800 1,3076000213623000 3,9368000030517600 2,0323998928070100
Particle 15
0,4352000057697300 1,4140000343322800 3,0587999820709200 2,7620000839233400
Particle 16
0,2331999987363820 0,9947999715805050 1,1468000411987300 1,8375999927520800
Particle 17
3,0759999752044700 2,3815999031066900 0,5824000239372250 2,2948000431060800

Particle 18
2,9695999622345 2,94079995155334 2,973200008277893 1,018000000667572
Particle 19
3,13039994239807 0,4711999893188 48 0,7148000001907 35 1,162400007247 92
Particle 20
0,7508000135421 75 2,089200019836 43 0,0883999988436699 1,009600043296 81
========== Velocity Awal ==========
Velocity untuk particle 1
2,53279995918274 -0,741599977016449 -3,776000002288818 2,028000116348 27
Velocity untuk particle 2
0,8392000198364 26 -3,967200040817 26 -2,140799999237 06 -0,6424000263214 11
Velocity untuk particle 3
1,212000012397 77 -0,613600015640259 0,875199973583221 -0,00319999999191612
Velocity untuk particle 4
-0,696799993515015 -0,5672000005054474 0,0983999967575073 -3,157599925994 87
Velocity untuk particle 5
0,3592000007629 39 -3,847199916839 6 3,665600061416 63 3,413599967956 54
Velocity untuk particle 6
0,5031999945640 56 3,743999958038 33 1,720800004215 24 -3,883199930191 04
Velocity untuk particle 7
1,111199975013 73 -3,085599899291 99 3,454400062561 04 -3,297600030899 05
Velocity untuk particle 8
1,84560000896454 0,7111999988555 91 -3,871200084686 28 -3,171999931335 45
Velocity untuk particle 9
-2,692800045013 43 -1,205600023269 65 -0,6007999777793 88 -0,9735999703407 29
Velocity untuk particle 10
0,9007999897003 17 -3,471999883651 73 -3,322400093078 61 -1,866400003433 23
Velocity untuk particle 11
3,931200027465 82 1,150400042533 87 0,0920000000178 8139 -1,167199969291 69
Velocity untuk particle 12
1,126399993896 48 3,635999917984 01 0,2768000066280 36 1,468000054359 44
Velocity untuk particle 13
2,028800010681 15 2,295200109481 81 -1,672000050544 74 -1,259199976921 08
Velocity untuk particle 14
-0,918399989604 95 2,032000064849 85 1,419999957084 66 -0,7864000201225 28
Velocity untuk particle 15
-1,320799946784 97 3,222399950027 47 -3,967999935150 15 2,237600088119 51
Velocity untuk particle 16
2,357599973678 59 -2,051199991302 49 -1,802399992942 81 0,3488000035285 95
Velocity untuk particle 17
1,101600050926 21 2,687999963760 38 3,415999889373 78 -1,962399959564 21
Velocity untuk particle 18
0,0136000001803041 0,3431999981403 35 -0,8503999710083 01 -3,891999959945 68
Velocity untuk particle 19
-0,7472000122070 31 1,301599979400 63 -2,723999977111 82 2,118400096893 31
Velocity untuk particle 20
3,421600103378 3 -2,347199916839 6 -3,998399972915 65 -3,139199972152 71
========== Permutasi Job dan Evaluasi ==========
---Dengan Menggunakan SPV Rule---
Posisi 1
Job Sequence : 3 1 4 2 => 76
Posisi 2
Job Sequence : 1 4 3 2 => 78
Posisi 3
Job Sequence : 3 1 2 4 => 75
Posisi 4
Job Sequence : 4 1 3 2 => 78
Posisi 5
Job Sequence : 4 2 1 3 => 64

Posisi 6
Job Sequence : 4 2 1 3 => 64
Posisi 7
Job Sequence : 4 1 2 3 => 64
Posisi 8
Job Sequence : 4 1 2 3 => 64
Posisi 9
Job Sequence : 4 3 2 1 => 77
Posisi 10
Job Sequence : 2 3 4 1 => 63
Posisi 11
Job Sequence : 4 1 2 3 => 64
Posisi 12
Job Sequence : 2 4 1 3 => 64
Posisi 13
Job Sequence : 1 3 2 4 => 77
Posisi 14
Job Sequence : 2 1 4 3 => 64
Posisi 15
Job Sequence : 3 4 2 1 => 81
Posisi 16
Job Sequence : 1 2 4 3 => 64
Posisi 17
Job Sequence : 2 1 4 3 => 64
Posisi 18
Job Sequence : 2 4 3 1 => 63
Posisi 19
Job Sequence : 3 4 2 1 => 81
Posisi 20
Job Sequence : 4 1 3 2 => 78
========== Personal Best ==========
Particle 1
2,0064001083374 3,730000001907349 1,42879998683929 2,55040001869202
Particle 2
2,71399998664856 0,8191999793052670 0,5483999848365780 1,61199998855591
Particle 3
3,26239991188049 1,722000000286102 3,361200094223020 2,28119993209839
Particle 4
1,90439999103546 3,79959988594055 2,152400001678467 3,67440009117126
Particle 5
0,399199992418289 2,267999887466430 0,925199985504150 0,106799997389317
Particle 6
3,553200006484990 2,690799951553340 1,800799965858460 0,086000002920627 6
Particle 7
3,691600084304810 0,662000000476837 2,869999885559080 0,628799974918365
Particle 8
3,141200065612790 2,923199892044070 1,769199967384340 0,358399987220764
Particle 9
1,696799993515010 2,664000034332280 2,694799900054930 1,373999953269960
Particle 10
3,854000091552730 0,064000003039836 9 1,758800029754640 2,065999984741210
Particle 11
1,420799970626830 1,169999957084660 3,769599914550780 2,556400060653690
Particle 12
2,348799943923950 0,547599971294403 2,174000024795530 1,105599999427800
Particle 13
2,802799940109250 0,680800020694733 3,444400072097780 3,288399934768680
Particle 14
2,769599914550780 1,307600021362300 3,936800003051760 2,032399892807010

Particle 15
0,43520000576973 1,414000003433228 3,058799998207092 2,762000008392334
Particle 16
0,2331999998736382 0,994799971580505 1,146800004119873 1,83759999275208
Particle 17
3,07599997520447 2,38159990310669 0,5824000023937225 2,294800004310608
Particle 18
2,9695999622345 2,94079995155334 2,97320008277893 1,01800000667572
Particle 19
3,13039994239807 0,4711999893188848 0,7148000000190735 1,16240000724792
Particle 20
0,7508000013542175 2,08920001983643 0,08839999884366699 1,009600004329681
========== Global Best ==========
3,85400009155273 0,06400000303983691 1,758800002975464 2,065999984741211

xxxxxxxxxxxxx Iterasi 10 xxxxxxxxxxxxxxx
========== Velocity ==========
Velocity untuk particle 1
-1,3699743416259 1,29156285555856 1,06681558383831 -1,577482502158511
Velocity untuk particle 2
0,5256960583339491 -1,6146651751316 2,8682440867426 -1,1751795545564451
Velocity untuk particle 3
-0,8180876577223941 -0,6382487619507741 0,2282178368421911 0,8816718506642471
Velocity untuk particle 4
-2,577998346735371 0,7121144444969081 0,05421948930241971 -0,008362889762935311
Velocity untuk particle 5
0,6151953539224831 -0,2005985810247781 1,258237275157211 -0,9973581991149351
Velocity untuk particle 6
0,1908529998129891 1,637569222328911 -1,157326568637051 -1,282322750857111
Velocity untuk particle 7
-3,078147291492721 2,415236593161471 -2,760431079059881 0,9598529668186441
Velocity untuk particle 8
1,677470477243071 0,6202002217289681 -1,48442999135831 -1,66710527050171
Velocity untuk particle 9
-0,7394918776412641 2,915451180611661 0,3989928833057661 2,962498729023121
Velocity untuk particle 10
0,2261746924555151 2,255289936187961 1,883612921637011 0,3794666628556631
Velocity untuk particle 11
1,8758918733375341 1,117291742405691 3,300265566275581 -0,2022512034275931
Velocity untuk particle 12
1,9055235721561441 0,4933349945376931 -0,04441989575753971 1,0193054531573911
Velocity untuk particle 13
3,3894917439286271 -0,5821652193697951 0,63337037405051 -1,739328863209611
Velocity untuk particle 14
-1,7629002880922721 -2,997842818046561 -0,8851192163319581 -2,0467895373277471
Velocity untuk particle 15
-1,6769919378333161 -2,064274164564351 0,9094564459445421 1,0130906549653371
Velocity untuk particle 16
-0,9443816778862411 1,014065375966271 1,232762798677531 2,3703030666446161
Velocity untuk particle 17
0,5476991616781961 -1,475857343740541 0,614412910030951 -1,756485885249591
Velocity untuk particle 18
-2,7399758100539471 -2,741524053556471 0,07653509841914821 -0,2255445904021511
Velocity untuk particle 19
-0,5735840429213681 -1,697417925649461 0,1554025438528161 -3,1923629858315161
Velocity untuk particle 20
3,24460815551911 -3,183221061839391 -2,277940457890721 -3,3341967878659271
========== Populasi ==========
Particle 1

1,38120436668396 0,999639332294464 2,41006171703339 4,44650340080261
Particle 2
5,52016282081604 -4,1590164899826 1,581007480621 34 0,6836446523666 38
Particle 3
0,918463289737701 0,542709052562714 2,13116526603699 4,49893492460251
Particle 4
1,76171231269836 0,712810635566711 1,81534684076905 1,57023847009987
Particle 5
1,60467088222504 0,643767669796944 1,48116935789585 2,948314547538 76
Particle 6
4,815061450004 58 3,49443006515503 3,084787964820 86 -0,967647910118103
Particle 7
4,81375813484192 -3,25777220726013 4,4566638469696 -0,6141843199729 92
Particle 8
2,0456477701664 0,9188460707664 49 1,918430924415 59 3,761034369468 69
Particle 9
1,50433927774429 5,783296346664 43 2,894175648689 27 3,47031903266907
Particle 10
4,406199797987 94 0,933401346206665 2,536282062530 52 1,875801205635 07
Particle 11
6,52465748786926 0,538187623023987 2,19227194786072 0,601506456732 75
Particle 12
6,65635704994202 3,471685171127 32 1,801673401147 13 3,771809339523 32
Particle 13
7,97539067268372 1,911232948303 22 -0,438217401504517 -0,626888275146484
Particle 14
2,74487793445587 0,917817592620 85 4,181206941604 61 -0,316361904144287
Particle 15
3,24767112731934 0,507953882217407 -0,0388918519020081 4,069085121154 79
Particle 16
3,98060566186905 -1,894530415534 97 0,6193528175354 3,730780601501 46
Particle 17
5,58045452833176 0,377483129501343 5,079530358314 51 -0,755458354949951
Particle 18
1,99409103393555 1,417286634445 19 2,597156636416 91 0,756739959120 75
Particle 19
3,68909078836441 0,064350128173 8281 -0,476642251014709 3,160341978073 12
Particle 20
1,66363573074341 0,729909181594849 2,5096390247345 4,340019464492 8
========== Permutasi Job dan Evaluasi ==========
---Dengan Menggunakan SPV Rule---
Posisi 1
Job Sequence : 2 1 3 4 => 63
Posisi 2
Job Sequence : 3 2 1 4 => 76
Posisi 3
Job Sequence : 1 4 2 3 => 64
Posisi 4
Job Sequence : 2 1 3 4 => 63
Posisi 5
Job Sequence : 3 1 4 2 => 76
Posisi 6
Job Sequence : 3 1 4 2 => 76
Posisi 7
Job Sequence : 4 3 1 2 => 76
Posisi 8
Job Sequence : 1 3 4 2 => 82
Posisi 9
Job Sequence : 2 3 1 4 => 62

Posisi 10
Job Sequence : 1 2 3 4 => 63
Posisi 11
Job Sequence : 3 1 2 4 => 75
Posisi 12
Job Sequence : 1 2 3 4 => 63
Posisi 13
Job Sequence : 1 2 3 4 => 63
Posisi 14
Job Sequence : 1 3 2 4 => 77
Posisi 15
Job Sequence : 3 2 4 1 => 76
Posisi 16
Job Sequence : 4 2 3 1 => 63
Posisi 17
Job Sequence : 4 3 2 1 => 77
Posisi 18
Job Sequence : 4 2 1 3 => 64
Posisi 19
Job Sequence : 2 1 4 3 => 64
Posisi 20
Job Sequence : 4 3 1 2 => 76
========== Personal Best ==========
Particle 1
2,020930767059330,559126555919647 1,978052675724035,9573438167572 <-- 2,020930767059330,559126555919647 1,978052675724035,9573438167572
Particle 2
5,147588253021243,180800020694730,403631031513214 1,702218793332585 <-- 5,147588253021243,180800020694730,403631031513214 1,702218793332585
Particle 3
0,6128229498863220,399118423461914 1,539256334304814,835250973701480 <-- 0,6128229498863220,399118423461914 1,539256334304814,835250973701480
Particle 4
2,92893457412721,113488793373111,83890892565251,963956832885740 <-- 2,92893457412721,113488793373111,83890892565251,963956832885740
Particle 5
0,8863782882690431,118676066398620,385386228561401 3,185793261975050 <-- 0,8863782882690431,118676066398620,385386228561401 3,185793261975050
Particle 6
4,4489178657533172,193224847316743,287687659263610,4939238056540490 <-- 4,4489178657533172,193224847316743,287687659263610,4939238056540490
Particle 7
7,178211688995365,266581296920786,699115514755250,8591609001159700 <-- 7,178211688995365,266581296920786,699115514755250,8591609001159700
Particle 8
0,7808511257171630,934674501419067 3,117070913314824,954559326171880 <-- 0,7808511257171630,934674501419067 3,117070913314824,954559326171880
Particle 9
1,5043392777444295,783296346664433,894175648689273,470319032669070 <-- 1,3569825887680144,303314208984383,562949419021611,943297624588010
Particle 10
3,8540000915527300,064000003039836 1,7588000297546402,065999984741210 <-- 3,8540000915527300,064000003039836 1,7588000297546402,065999984741210
Particle 11
5,4207999706268300,5771067738533020,8921003341674800,7839530706405640 <-- 5,4207999706268300,5771067738533020,8921003341674800,7839530706405640
Particle 12
5,578515529632573,108043849468231,811862587928773,840700984001160 <-- 5,578515529632573,108043849468231,811862587928773,840700984001160

Particle 13
6,17629671096802 1,83842700719833 -0,541940450668335 0,355502843856812 <-- 6,17629671096802 1,83842700719833 -0,541940450668335 0,355502843856812
Particle 14
3,97472906112671 2,46170526742935 4,57959008216858 1,03942346572876 <-- 3,97472906112671 2,46170526742935 4,57959008216858 1,03942346572876
Particle 15
3,82497549057007 1,12254476547241 -0,217753171920776 4,05850201845169 <-- 3,82497549057007 1,12254476547241 -0,217753171920776 4,05850201845169
Particle 16
4,23319999873638 -2,22160375118256 0,425626516342163 2,48777043819427 <-- 4,23319999873638 -2,22160375118256 0,425626516342163 2,48777043819427
Particle 17
5,21281170845032 1,38882917165756 4,58240002393723 0,1918005943298 34 <-- 5,21281170845032 1,38882917165756 4,58240002393723 0,1918005943298 34
Particle 18
2,9695999622345 2,94079995155334 2,97320008277893 1,01800000667572 <-- 2,9695999622345 2,94079995155334 2,97320008277893 1,01800000667572
Particle 19
3,52320411801338 1,04316008090973 -0,199823021888733 4,39924001693726 <-- 3,52320411801338 1,04316008090973 -0,199823021888733 4,39924001693726
Particle 20
0,0369453430175781 2,35502171516418 3,93424880504608 6,02463483810425 <-- 0,0369453430175781 2,35502171516418 3,93424880504608 6,02463483810425
========== Global Best ==========
2,06599998474121 0,0640000030398369 1,75880002975464 3,85400009155273
========== Local Search ==========
@@@@@ Prosedur Local Search VNS @@@@@
Job Sequence Awal : 2 3 1 4
-----Fungsi Insert-----
Kappa : 3
Tetha : 4
Job Sequence : 2 3 1 4
Makespan : 62
Looping ke- 0
-----Fungsi Insert-----
Kappa : 3
Tetha : 2
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange-----
Kappa : 4
Tetha : 3
Job Sequence : 2 3 4 1
Makespan : 63
Looping ke- 1
-----Fungsi Insert-----
Kappa : 4
Tetha : 3
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange-----
Kappa : 2
Tetha : 3
Job Sequence : 2 1 3 4
Makespan : 63
Looping ke- 2
-----Fungsi Insert-----
Kappa : 4

Tetha : 3
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange------
Kappa : 1
Tetha : 4
Job Sequence : 4 3 1 2
Makespan : 76
Looping ke- 3
-----Fungsi Insert-----
Kappa : 2
Tetha : 1
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange------
Kappa : 4
Tetha : 3
Job Sequence : 2 3 4 1
Makespan : 63
Looping ke- 4
-----Fungsi Insert-----
Kappa : 4
Tetha : 1
Job Sequence : 3 1 2 4
Makespan : 75
-----Fungsi Interchange------
Kappa : 1
Tetha : 4
Job Sequence : 4 3 1 2
Makespan : 76
Looping ke- 5
-----Fungsi Insert-----
Kappa : 4
Tetha : 3
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange------
Kappa : 2
Tetha : 3
Job Sequence : 2 1 3 4
Makespan : 63
Looping ke- 6
-----Fungsi Insert-----
Kappa : 2
Tetha : 3
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange------
Kappa : 1
Tetha : 3
Job Sequence : 1 3 2 4
Makespan : 77
Looping ke- 7
-----Fungsi Insert-----
Kappa : 4
Tetha : 2
Job Sequence : 2 1 3 4
Makespan : 63
-----Fungsi Interchange------

Kappa : 1
Tetha : 2
Job Sequence : 3 2 1 4
Makespan : 76
Looping ke- 8
-----Fungsi Insert-----
Kappa : 2
Tetha : 4
Job Sequence : 2 3 4 1
Makespan : 63
-----Fungsi Interchange------
Kappa : 4
Tetha : 2
Job Sequence : 2 4 1 3
Makespan : 64
Looping ke- 9
-----Fungsi Insert-----
Kappa : 2
Tetha : 3
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange------
Kappa : 4
Tetha : 1
Job Sequence : 4 3 1 2
Makespan : 76
Looping ke- 10
-----Fungsi Insert-----
Kappa : 2
Tetha : 3
Job Sequence : 2 3 1 4
Makespan : 62
-----Fungsi Interchange------
Kappa : 3
Tetha : 4
Job Sequence : 2 3 4 1
Makespan : 63
Looping ke- 11
-----Fungsi Insert-----
Kappa : 1
Tetha : 4
Job Sequence : 2 4 3 1
Makespan : 63
-----Fungsi Interchange------
Kappa : 4
Tetha : 1
Job Sequence : 4 3 1 2
Makespan : 76
*******Hasil dari Local Search VNS*******
Global Best : 2,06599998474121 0,06400000030398369 1,758800002975464 3,85400009155273
Job Sequence : 2 3 1 4
Makespan : 62

Lampiran 4 : Hasil Implementasi Program Dengan Parameter Yang Berbeda

1. Kasus 20-*Job* 5-Mesin

a. $w^0 = 0{,}1$ dan $\alpha = 0{,}2$

```
***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN
PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job     : 20
Jumlah Mesin   : 5
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1     : 2
C2     : 2
wo     : 0,1
Alpha  : 0,2
=== Solusi yang diperoleh ===
Job Sequence :  9 15 6 3 14 8 4 16 17 5 18 7 11 10 19 1 2 13 20 12
Makespan     :  1278
```

b. $w^0 = 0{,}3$ dan $\alpha = 0{,}6$

```
***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN
PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job     : 20
Jumlah Mesin   : 5
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1     : 2
C2     : 2
wo     : 0,3
Alpha  : 0,6
=== Solusi yang diperoleh ===
Job Sequence :  17 3 15 6 11 9 13 7 19 8 5 18 14 16 4 2 1 10 20 12
Makespan     :  1278
```

c. $w^0 = 0{,}4$ dan $\alpha = 0{,}2$

```
***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN
```

PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 5
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
c1 : 2
c2 : 2
wo : 0,4
Alpha : 0,2
=== Solusi yang diperoleh ===
Job Sequence : 9 15 6 19 14 13 3 1 17 5 4 16 18 8 2 7 11 10 20 12
Makespan : 1278

d. w^0 = 0,6 dan α = 0,2

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 5
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
c1 : 2
c2 : 2
wo : 0,6
Alpha : 0,2
=== Solusi yang diperoleh ===
Job Sequence : 17 3 15 6 19 14 9 4 5 11 13 18 7 16 1 8 2 10 20 12
Makespan : 1278

e. w^0 = 0,9 dan α = 0,1

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 5
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===

C1 : 2
C2 : 2
wo : 0,9
Alpha : 0,1
=== Solusi yang diperoleh ===
Job Sequence : 3 17 15 6 9 14 7 11 19 13 1 8 5 2 4 18 16 10 20 12
Makespan : 1278

2. Kasus 20-*Job* 10-Mesin

a. $w^0 = 0,1$ dan $\alpha = 0,5$

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 10
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1 : 2
C2 : 2
wo : 0,1
Alpha : 0,5
=== Solusi yang diperoleh ===
Job Sequence : 18 5 2 12 9 10 15 17 4 19 3 6 14 8 20 11 13 7 1 16
Makespan : 1586

b. $w^0 = 0,2$ dan $\alpha = 0,7$

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 10
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1 : 2
C2 : 2
wo : 0,2
Alpha : 0,7
=== Solusi yang diperoleh ===
Job Sequence : 18 5 2 12 9 10 15 4 17 19 3 6 14 8 20 11 13 7 1 16
Makespan : 1586

c. w^0 = 0,5 dan α = 0,7

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 10
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1 : 2
C2 : 2
wo : 0,5
Alpha : 0,7
=== Solusi yang diperoleh ===
Job Sequence : 5 9 12 17 15 3 4 18 2 8 19 10 6 14 20 11 13 7 1 16
Makespan : 1586

d. w^0 = 0,6 dan α = 0,1

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20
Jumlah Mesin : 10
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
C1 : 2
C2 : 2
wo : 0,6
Alpha : 0,1
=== Solusi yang diperoleh ===
Job Sequence : 5 9 12 17 15 3 4 18 2 8 13 10 6 19 11 14 20 7 1 16
Makespan : 1587

e. w^0 = 0,9 dan α = 0,3

***** Aplikasi C++ Builder *****
ALGORITMA PSO DENGAN LOCAL SEARCH UNTUK PERMASALAHAN PENJADWALAN PERMUTATION FLOWSHOP
=== Permasalahan yang akan diselesaikan ===
Jumlah Job : 20

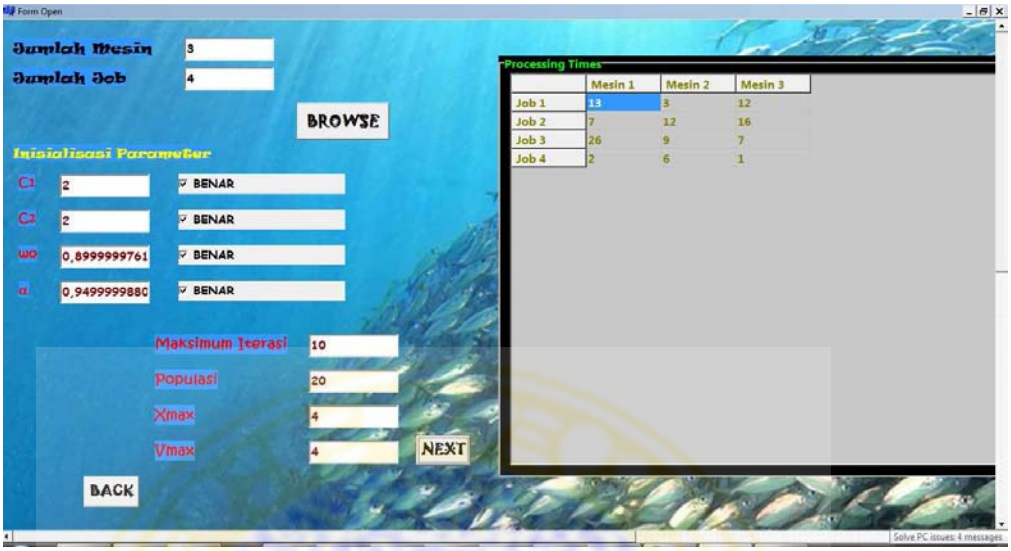
Jumlah Mesin : 10
Jumlah Iterasi : 10
Jumlah Particle : 20
Xmax : 4
Vmax : 4
=== Parameter yang digunakan ===
c1 : 2
c2 : 2
w0 : 0.9
Alpha : 0.3
=== Solusi yang diperoleh ===
Job Sequence : 5 9 12 17 15 3 18 4 2 8 19 10 6 14 20 11 13 7 1 16
Makespan : 1586

Lampiran 5 : OutputProgram

1) Tampilan Awal Program

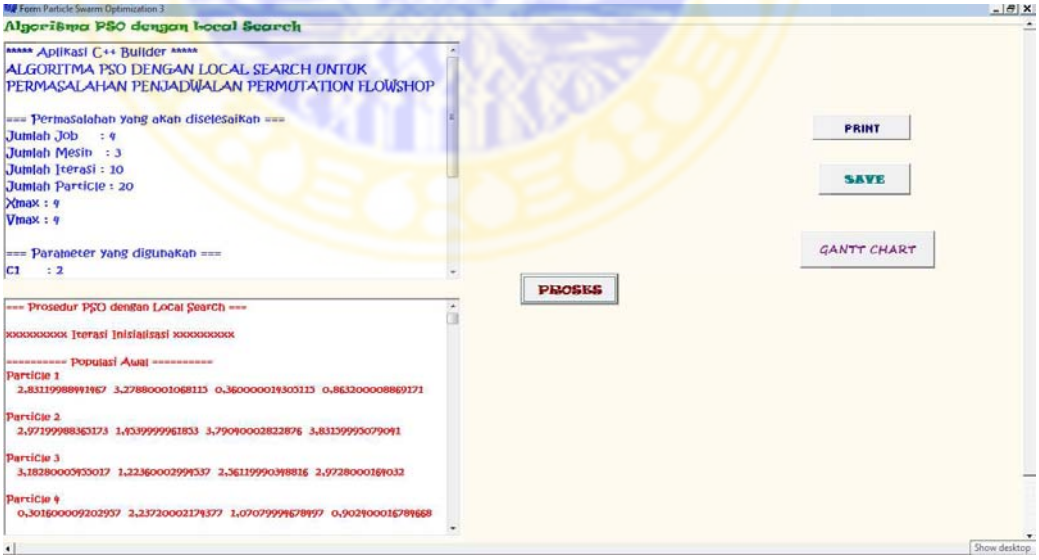
Merupakan tampilan awal ketika hendak menggunakan program. Form ini berisikan pilihan untuk menjalankan program, seperti pilihan untuk memasukkan permasalahan baru **Form→New Problem** atau pilihan untuk membuka file dengan data yang sudah ada **Form→Open File**, berbeda dengan **Open File** untuk pilihan **Form→Open Data** berisikan file dengan data yang sudah disimpan sebelumnya namun data yang ada didalamnya tidak memuat nilai-nilai parameter yang dibutuhkan.

2) *Input* Awal Permasalahan dan Parameter



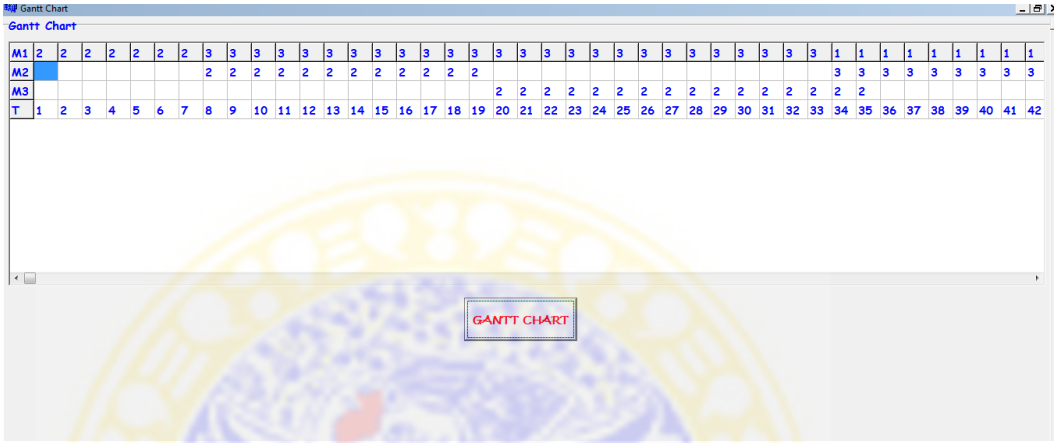
Tampilan untuk *input* awal permasalahan dan parameter. Antara lain, jumlah mesin, jumlah *job*, parameter-parameter seperti *social* parameter (c_1), *cognitive* parameter (c_2), *inertia weight* (w), dan *decrement factor* (α).

3) Hasil Serta Proses Pengerjaan



Berisikan hasil akhir yang didapatkan dari permasalahan yang diselesaikan. Selain itu pada form ini juga berisi mengenai proses jalannya algoritma PSO dengan *local search* hingga ditemukan solusi terakhir.

4) *Gantt Chart*



Pada form ini, solusi akhir yang didapatkan kemudian ditampilkan dalam bentuk *gantt chart*. *Job-job* yang sesuai dengan jadwal yang didapatkan kemudian disusun, misal pada mesin 1, *job* 2 dikerjakan terlebih dahulu dengan waktu pengerjaan sebesar 7 satuan waktu, maka pada *gantt chart* untuk mesin 1 akan diisi dengan inisial dari *job* 2 sebanyak 7 kotak sesuai dengan *processing time* untuk *job* 2 pada mesin 1, begitu juga pengisian untuk *job-job* yang lain pada masing-masing mesin.